LA SCOPERTA DEL COMMODORE 64

1. introduzione al basic

daniel - jean david

La scoperta del COMMODORE 64

1. Introduzione al BASIC

DELLO STESSO EDITORE

Volumi pubblicati

D.-J. David - La pratica del Commodore 64
X. Linant de Bellefonds - La pratica dello ZX Spectrum - Vol. 1
M. Henrot - La pratica dello ZX Spectrum - Vol. 2
J.-F. Séhan - Chiavi per lo ZX Spectrum
C. Galais - Vademecum per Applesoft
J. Boisgontier - L'Apple e i suoi files
B. De Merly - Guida per l'Apple

Volumi di prossima pubblicazione

J. Boisgontier - 102 programmi per il Commodore 64
J. Boisgontier - Commodore 64: metodi pratici
J. Boisgontier, S. Brebion, G. Foucault - Il Commodore 64 per tutti
B. De Merly - Guida per l'Apple - Vol. 2
B. De Merly - Guida per l'Apple - Vol. 3
F. Lévy - Esercizi per l'Apple II
J. Boisgontier - 36 programmi su Apple II per tutti
F. Lévy - Esercizi per lo ZX Spectrum
J.-F. Séhan - Alla ribalta: lo ZX Spectrum
J. Deconchat - 102 programmi per ZX Spectrum e ZX 81
D.A. Lien - Dizionario del Basic
J. Boisgontier - Il Basic per tutti
A. Pinaud - CP/M passo dopo passo

DANIEL-JEAN DAVID

La scoperta del COMMODORE 64

1. Introduzione al BASIC

Edizione italiana a cura di
FRANCO POTENZA

Traduzione di
ANDREA PANAGIA

Editsi - Editoriale per le scienze informatiche - S.r.l.
MILANO 1984

Daniel-Jean David *insegna informatica gestionale all'Università di Parigi - 1 Panthéon Sorbonne.*

Insegna inoltre l'uso dei microprocessori all'ENSAM.

I campi della sua ricerca vanno dall'informatica grafica alle tecniche di interfacciamento dei microprocessori e dei sistemi multiprocessori.

Specialista del microprocessore 6502, ha tenuto, a Parigi, numerosi seminari sui microprocessori, KIM, SYM e PET.

È direttore del "La Commode", rivista trimestrale francese rivolta ai calcolatori Commodore.

Titolo originale dell'opera
La découverte du COMMODORE 64
Initation au Basic

Presentazione

Questo libro ha per scopo di permettervi di trarre il massimo possibile dal vostro microcomputer C64. Dopo una introduzione composta da richiami generali sull'informatica, comprende essenzialmente una introduzione progressiva al linguaggio Basic, che è il linguaggio di programmazione più spesso utilizzato sul C64.
Ovviamente vi si sfruttano al massimo le particolarità del C64. La struttura di tale parte è concepita per permettere l'acquisizione progressiva delle conoscenze: essa è formata da capitoli, o piuttosto da serie.
In ogni serie viene costruito pezzo per pezzo un programma, per variazioni continue, introducendo a poco a poco le nozioni nuove.
Raccomandiamo al lettore di seguire tale progressione, di provare effettivamente sul proprio C64 le diverse versioni dei programmi, e pure di immaginarne altri.
È la condizione necessaria per l'acquisizione di nozioni durevoli. Tuttavia, questo libro dovrebbe anche permettere alle persone che non hanno ancora un C64 di farsi un'idea delle possibilità di questo computer.
Per finire, l'opera termina con delle appendici dove vengono fornite informazioni di referenza: spiegazione di ogni istruzione Basic, messaggi d'errore, punti particolari trattati sotto forma di domande e di risposte.
Ultima precisazione: tutti i programmi citati sono stati realmente provati alla tastiera di un C64.

Sommario

Capitolo 1

Presa di contatto

IL MESSAGGIO "38911 BASIC BYTES FREE"

Bits, bytes, informazioni

Vi siete appena seduti di fronte al vostro C64 e l'avete appena acceso, ed ecco che visualizza "38911 BASIC BYTES FREE" la cui traduzione è: "38911 BYTES LIBERI IN BASIC". Ma cosa sono questi "BYTES" dei quali vi si annuncia che sono "LIBERI"?
Per capire — e ce ne scusiamo — dobbiamo esaminare alcune nozioni. Cercheremo di essere brevi. Il C64 è un computer, cioè una macchina per il trattamento automatico di informazioni. Si capisce che i computer abbiano un campo di applicazioni estremamente vasto se si pensa che, in definitiva, ogni attività umana si riconduce ad un certo trattamento d'informazioni.
Una delle operazioni essenziali, che il computer deve poter effettuare su di una informazione, è memorizzarla per essere capace di utilizzarla in vari momenti. Ma la memorizzazione dell'informazione implica la sua riduzione ad una forma fisica perché possa venir immagazzinata nei circuiti del computer.
Allo stato attuale della tecnologia, l'unica codifica pratica è di tipo binario, poiché si sanno utilizzare benissimo elementi capaci di acquisire due stati ben distinti, ad esempio: presenza od assenza di un foro su di una scheda, elemento di banda magnetica magnetizzato in un senso o nell'altro, elemento di circuito elettrico elevato al potenziale di 5 V, o lasciato a 0, ecc.
Tale elemento che, in quanto informazione, è in pratica capace di contenere la risposta per si o per no ad una certa domanda si chiama BIT (abbreviazione inglese di "cifra binaria": Bynary Digit).
Ma un solo bit costituisce in genere un'informazione troppo elementare per poterla manipolare in modo pratico. Perciò si manipolano generalmente gruppi di bits. Il gruppo più spesso considerato è il

gruppo di 8 bits o BYTE. Per capire cosa significa "averne 38911 liberi", dobbiamo ora vedere quello che si può rappresentare con un byte, cioè quali informazioni si possono racchiudere in un byte.
Per visualizzare un byte sulla carta, dobbiamo introdurre due simboli corrispondenti ai due stati che può assumere ogni bit del byte. Se scegliamo come simboli 0 e 1, un byte potrà essere, ad esempio, 01000001 o 10100101 oppure 01101001. Constatiamo subito che, avendo due possibilità per ogni bit, abbiamo $2^8 = 256$ possibilità differenti per un byte.
Ora, se vogliamo che un byte rappresenti un numero, ciò è molto facile. Basta considerare che il numero è stato espresso nel sistema di numerazione binaria (di base due). Così ad esempio, 01000001 varrà $1 + 0 \times 2 + 0 \times 2^2 + 0 \times 2^3 + 0 \times 2^4 + 0 \times 2^5 + 1 \times 2^6 + 0 \times 2^7 = 1 + 64 = 65$. Così come in decimale 1702 vale $2 + 0 \times 10 + 7 \times 10^2 + 1 \times 10^3$. Ciascun bit rappresenta una cifra del sistema binario, dal che si deriva il suo nome. Il più piccolo numero che si possa rappresentare è 00000000 (0); il più grande è 11111111 (255): ritroviamo le 256 combinazioni. Vediamo anche una cosa: siccome vogliamo poter manipolare numeri più grandi di 255, talvolta i numeri dovranno occupare più bytes.
Si può riporre qualcosa d'altro che dei numeri in un byte? Naturalmente! Supponiamo di decidere che 01000001 = A, 01000010 = B ecc. per tutte le lettere. Di nuovo, si può avere una serie di 256 caratteri diversi, il che permette le lettere maiuscole e minuscole, le cifre, i caratteri di punteggiatura e tanti altri. Dunque potremo immagazzinare qualsiasi testo nella memoria, a ragione di un carattere per byte.
Quindi si può riporre un testo di 39000 caratteri nella memoria del C64, ossia 20 pagine di questo libro. Ciò vi permette, ora, di farvi un'idea migliore di cosa abbiamo come memoria.
Ma c'è ancora un'altra categoria di informazioni che devono entrare nella memoria, ed è pure una caratteristica fondamentale dei computers. Infatti un computer necessita, per funzionare, di istruzioni che gli dicano cosa deve fare. Queste istruzioni codificate opportunamente risiedono in memoria alla stregua delle informazioni da trattare; analogamente, un impiegato che effettua calcoli di contabilità non ha forse in memoria la lista delle operazioni che deve effettuare accanto alle cifre che deve manipolare?

La configurazione del C64

La memoria non è altro che una parte della configurazione di cui disponiamo con il C64. Sui grossi computers si distinguono facilmente, giacché occupano ciascuno un armadio, gli elementi principali che sono l'unità centrale (dove si effettuano i trattamenti) e le periferiche che servono a comunicare con il mondo esterno (segnatamente cogliere i dati da trattare e fornire i risultati ottenuti).

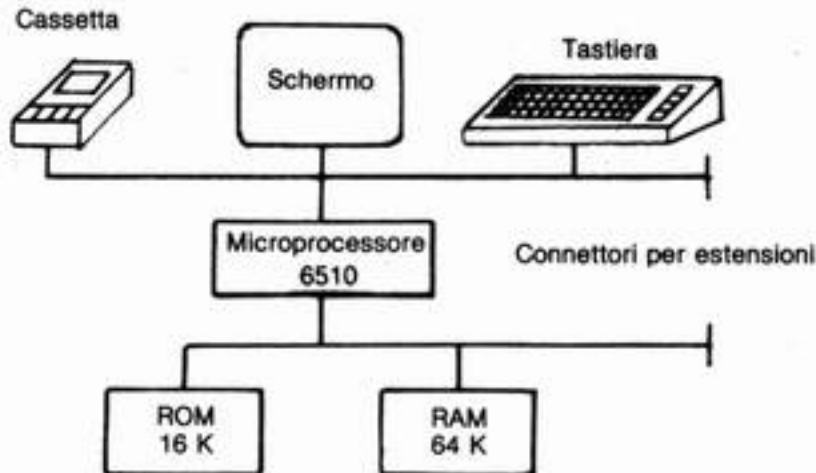



Fig. 1.1. Sinottica del C64.

Gli stessi elementi esistono sul C64. Esteriormente, si vedono solo le periferiche:

— la *tastiera*, che ci servirà ad inserire dati ed istruzioni;
— lo *schermo televisivo*, sul quale saranno visualizzati i risultati (25 linee di 40 caratteri). La coppia tastiera/schermo è lo strumento del dialogo tra voi e la vostra macchina;
— l'*unità di cassette magnetiche*, a differenza delle periferiche di comunicazione precedenti, si tratta piuttosto di una periferica di stoccaggio a memoria di massa, che permette di archiviare dei programmi o dei dati se la memoria centrale è insufficiente. Ma la cassetta è pure una periferica di comunicazione tra C64: voi potete inviare un programma o dei dati su cassetta a un amico, che possiede pure un C64;
— l'*unità centrale* è nascosta, ma esiste ed in pratica è formata da due elementi: il *processore* e la *memoria centrale*;
— il *processore* qui è un *microprocessore*, cioè un circuito integrato su grande scala capace di comandare tutto il sistema da solo: cerca in memoria le istruzioni successive, le interpreta e le esegue; manda, agli altri componenti del sistema, gli ordini necessari. Nel caso del C64 il microprocessore è un 6510 di Tecnologia MOS, uno dei più efficienti del mercato;
— la *memoria*. I 38911 bytes visti prima, non ne formano che una parte. In realtà sono 82000 i bytes di memoria presenti [1]; (gli informatici dicono 82 K dove $K = 2^{10} = 1024$). Degli 82 K, 18 K sono *ROM* (Read Only Memory), memoria scritta una volta per tutte pure chiamata Memoria Morta o *MEM*, che contiene quei programmi invaria-

(1) Ma non accessibili simultaneamente. I 64 K bytes di memoria viva ("64 K RAM SYSTEM") sono, in parte, occultati dalla memoria morta, perché la capacità di indirizzamento del microprocessore è limitata a 64 K.

bili che permettono al C64 di mettersi al servizio dell'utente (è il sistema di gestione che presentiamo più dettagliatamente nella prossima sezione). I 40 K rimanenti sono *RAM* (Random Access Memory), cioè memoria che si può leggere o scrivere (o Memoria Viva, *MEV*). Essa contiene i programmi dell'utente ed i dati variabili. Il C64 ne riserva 2 K per suo uso (un K per lo schermo, un K di memoria di lavoro per il sistema), dunque ne rimangono 38 K liberi per l'utente.
La figura 1.1 dà una veduta sinottica della configurazione del C64.
Abbiamo, ormai, completamente interpretato "38911 BYTES FREE".
Adesso dobbiamo spiegare la linea "CBM BASIC V2" che la precede.
Sarà lo scopo della sezione successiva.

Programmi - Sistema di gestione - BASIC

Abbiamo appena visto che il microprocessore, elemento chiave dell'unità centrale, è capace di cercare in memoria le *istruzioni* successive, di decifrale e di obbedir loro. In realtà, non è capace d'altro! Poi, per ottenere qualunque cosa dal computer, bisognerà fornigli le istruzioni appropriate. Senta precise istruzioni, il computer non sa fare nulla, non ha alcuna iniziativa.
Una sequenza di istruzioni che il computer deve eseguire in successione si chiama un *programma*. Fornire una tale sequenza di istruzioni che permetta di risolvere un certo problema si chiama programmare.
Capite dunque che dovrete fornire programmi al vostro C64, li introdurrete mediante la tastiera. Ma, affinché il C64 tenga conto di ciò che batterete alla tastiera, deve avere un programma che glielo ordini. Da solo non avrebbe l'iniziativa di andare a vedere se qualcuno batte sulla tastiera...
Per fortuna, non dobbiamo scrivere questo programma. Infatti, il C64 — come ogni altro computer d'altra parte — viene consegnato con un insieme di programmi fondamentali che sono indispensabili per il suo uso: programma che legge la tastiera, programma che visualizza sullo schermo, programma che gestisce le cassette, programma che attende le istruzioni dell'utente, ecc. L'insieme di questi programmi si chiama il *sistema di gestione*.
Questi programmi risiedono in ROM, allo scopo di essere conservati in permanenza per essere disponibili appena accendiamo il C64. Il messaggio che appare sullo schermo appena è messo sotto tensione è un modo per dire all'utente che il sistema di gestione si mette a sua disposizione ed aspetta i suoi ordini.
Una componente molto importante del sistema di gestione è l'*interprete BASIC* che ci permetterà di fornire istruzioni al C64 sotto una forma per noi più comoda.
Infatti si pone il problema di sapere in che forma — o in quale linguaggio — dobbiamo fornire le nostre istruzioni al C64. A dire il ve-

ro, un computer non "capisce" che un solo linguaggio, il *linguaggio macchina* o *binario*.
Ad esempio, nel linguaggio macchina del C64, "sommare 2 e 3" si direbbe 10101001 00000010 00011000 01101001 00000011 10000101 01010000... è estremamente complicato da capire e da utilizzare per l'uomo! Per questo sono stati inventati altri linguaggi di programmazione, che si chiamano linguaggi evoluti e sono più vicini al linguaggio umano, più vicini alle notazioni matematiche usuali, e quindi più facili da usare. In tale linguaggio, "sommare 2 e 3" si direbbe ad esempio A = 2 + 3 è molto più comprensibile, non vi pare?
BASIC è uno di questi linguaggi evoluti, attualmente il più diffuso sui PSI (Piccoli Sistemi Individuali). È probabilmente il più semplice da usare per i principianti: il suo nome è l'abbreviazione di Beginners' All-purpose Symbolic Instruction Code = codice simbolico d'istruzioni di uso generale per i principianti.
È lui che utilizzeremo sul C64, ma non possiamo farlo senza l'aiuto di un programma del sistema di gestione. Infatti, il C64 non capisce realmente che il suo linguaggio macchina; è dunque necessario un programma del sistema di gestione che prenda ogni istruzione del nostro programma BASIC e la traduca in binario per eseguirla: questa è la funzione — essenziale — svolta dall'interprete BASIC.
Riassumendo, ciò che appare sullo schermo alla messa in funzione significa: l'interprete BASIC del sistema di gestione è a vostra disposizione — disponete per il vostro programma BASIC ed i vostri dati di 38911 bytes liberi. L'ultima parola "READY" che riapparirà ogni volta che il C64 avrà terminato un comando significa "Sono pronto ed aspetto che battiate la prossima istruzione".
Facciamolo...

I due modi di funzionamento di BASIC

Mettiamoci alla tastiera e battiamo:

BUONGIORNO C64 'Return'

(Ogni messaggio battuto dall'utente termina normalmente con il tasto 'Return' — ritorno carrello. Nel seguito, cesseremo progressivamente di farlo figurare: sarà sottinteso e dovrà essere battuto).
Il C64 risponde:

?SYNTA ERROR

Infatti, benché il nostro messaggio sia molto educato, non costituisce una corretta istruzione BASIC e, in conseguenza, il C64 (o piuttosto l'interprete BASIC) la rifiuta. Non bisogna crearsi complessi per i messaggi d'errore, ma ricercare con calma l'errore che si è fatto; il comportamento della macchina è, in tutti i casi, perfettamente razionale. Il C64 ha un repertorio di messaggi d'errore che non hanno altro scopo se non quello di aiutarci a correggerli.

Battiamo ora:

?"BUONGIORNO!SONO IL C64" 'Return'

(gli spazi si ottengono mediante la sbarra di spazio).
Questa volta, abbiamo battuto una buona istruzione, ed otteniamo per risposta:

BUONGIORNO! SONO IL C64
READY.

Non ottenete questa risposta? siete ben sicuri di aver battuto **tutti** i caratteri, comprese le virgolette?
Il punto interrogativo significa semplicemente "stampare". Constatiamo che abbiamo ottenuto una risposta immediata all'istruzione.
Parimenti, se battiamo:

?2+2 'Return'

il C64 farà il calcolo e stamperà immediatamente il risultato. Si ha quindi un modo di funzionamento pressappoco simile a quello di una calcolatrice tascabile, dove un'istruzione è eseguita appena è stata battuta. Si dice che si tratta del "modo immediato" o "modo d'esecuzione immediata" ovvero "modo diretto".
Il C64 ha un secondo modo di funzionamento. Battiamo:

20 ?2+2 'Return'

Non succede nulla. Battiamo ancora:

10 ?"BUONGIORNO" 'Return'

Ancora nulla. Battiamo:

RUN (le lettere R U N, non il tasto 'Run/stop')

Questa volta otteniamo sullo schermo:

BUONGIORNO
4
READY.

Cos'è successo? Ebbene, abbiamo funzionato nel secondo modo, nel quale le istruzioni non vengono eseguite immediatamente, ma sono messe in memoria per esecuzione ulteriore. Formano allora un programma che viene eseguito quando battiamo il comando RUN in modo diretto.
Tranne un paio di eccezioni, sono esattamente le stesse istruzioni che possono essere date nel modo diretto o nel secondo modo, chiamato "modo differito" o "modo programmato".
Allora, da cosa si riconosce il modo?
È semplicissimo: in modo differito, ogni istruzione possiede in testa un numero di linea, mentre in modo diretto non c'è numero di linea:

?2+2 modo diretto
10 ?2+2 modo differito (bisogna battere RUN per ottenere la risposta).

Oltre ad imporre il modo programmato, il numero di linea svolge un'altra funzione: vediamo dall'esempio precedente che l'istruzione 10 è stata eseguita prima dall'istruzione 20, benché sia stata battuta dopo: le istruzioni vengono eseguite non nell'ordine cronologico nel quale sono state battute, ma in ordine di numeri di riga crescenti.
Il C64 è naturalmente molto più usato in modo programmato, ma, prima, per familiarizzarci, effettueremo alcuni calcoli aritmetici in modo diretto.

Aritmetica in modo diretto

Come ogni calcolatrice, il C64 permette di valutare espressioni più complicate di 2 + 2! In ogni caso si deve incominciare con un ? o con la parola PRINT di cui è l'abbreviazione: ciò significa "stampate il risultato dell'espressione che segue".
Provate (i 'Return' sono sottintesi)

?5—3.5 (sottrazione; risultato 1.5)
?3*12 (moltiplicazione; risultato 36)
?1/3 (divisione; risultato .333333333)
?2↑5 (elevazione alla potenza; risultato $2^5 = 32$)

Si vede dunque quali sono i segni di operazione fondamentali. Si noti * e non x per la moltiplicazione, ↑ per l'elevazione alla potenza (da tastiera non si può mettere un numero più in alto di un altro).

Variabili

Data un'espressione, se ne può fare altro che stampare il suo valore: si può mettere il valore in memoria per uso ulteriore in un'altra espressione.
Per fare ciò, basta dare un nome all'espressione, battendo ad esempio:

A=2/3

Si dice che è stata costituita una variabile di nome A. Il C64 le attribuisce automaticamente un posto memoria (che non dovete conoscere: il C64 si occupa interamente della gestione della memoria). Poi, viene calcolato il risultato e viene riposto nella casella memoria considerata; non appare sullo schermo.
La variabile può ora essere usata in un'altra espressione; se battiamo:

?2*A

otteniamo

1.33333333

Quali nomi di variabili si possono prendere? I nomi di variabili sono praticamente arbitrari, con la limitazione che devono piegarsi alle seguenti costrizioni:

— *primo carattere:* lettera;
— *caratteri successivi:* lettere o cifre (es. A, Al, VAR, RESULT, H2SO4).

Il C64 permette più di due caratteri, ma non tiene conto che dei due primi. Se due variabili che considerate distinte hanno i loro due primi caratteri identici, il C64 le confonderà: ad esempio, è il caso di ALBERTO e ALDO.
Il fatto di poter usare più di due caratteri permette di scegliere nomi "evocativi" come RESULT, TASSO, CAP..., ma il nome non deve contenere una delle parole speciali del linguaggio BASIC, che si chiamano parole chiave: CIFRA è proibita perché contiene IF che è una parola di significato particolare per BASIC.
La lista delle parole chiave "riservate" è fornita in appendice.
L'uso di una variabile non ne distrugge il valore, o piuttosto non cancella la memoria corrispondente. Non è che in caso di una nuova istruzione di assegnamento (della forma A=...) che il valore verrà cambiato.
Provate il seguente dialogo (omettiamo i 'Return' ed i READY):

```
AL=3
?AL+5
8          (5+3=8)
?3*AL      (AL vale sempre 3)
9          (3×3=9)
ALDO=1     (ALDO è la stessa variabile di AL, ora ha preso il valore 1)
?AL+1
2          (1+1=2)
```

Rimane ancora un problema: a quante variabili abbiamo diritto? Il limite reale è di fatto la dimensione della memoria ma possiamo sperare di avere centinaia di variabili, cioè abbastanza da trattare i problemi più complessi.

Valutazione delle espressioni

Si possono calcolare espressioni più complesse che racchiudano più operazioni. Provate:

```
?5+4*2
?2*3↑2
?5*3/4
```

Il risultato della prima è 13, cioè è stata effettuata prima la moltiplicazione. La seconda dà 18, perché è stato effettuato prima il 3^2. La

terza si valuta calcolando prima 5×3 dividendo poi il risultato per 4. Riassumendo, si effettuano generalmente le operazioni da sinistra verso destra ma esiste una **regola di priorità** degli operatori:

↑ è il più prioritario, poi abbiamo:
— (prendere l'opposto, esempio: —X), poi:
* e / (ex-aequo), poi:
+ e — (ex-aequo).

Se si vuole cambiare l'ordine delle priorità, si usano parentesi: un gruppo tra parentesi è sempre valutato per primo. Ad esempio:

(2+10)/5 dà 2.4 mentre 2+10/5 dà 4
6—(3+5) dà —2 mentre 6—3+5 dà 8.

Si possono usare parentesi composte, ma deve sempre esserci lo stesso numero di parentesi aperte e chiuse e l'espressione deve sempre avere un senso. Ad esempio:

?(A—3*(B—C))/5↑(B*C)

Disponiamo, per facilitare i calcoli, di un insieme di funzioni matematiche che possono intervenire nelle espressioni aritmetiche. Il loro argomento, come ogni sottoespressione tra parentesi, viene valutato prioritariamente. La lista completa di tali funzioni è data in appendice, ma citiamo subito: SIN (seno), COS (coseno), EXP (esponenziale)

?1+SQR(2) dà 2.41421356

Precisione dei numeri

Esaminando i risultati ottenuti negli esempi precedenti, possiamo fare alcune osservazioni:

— Il C64 può stampare numeri positivi o negativi. Stampa uno spazio in testa per un numero positivo, un — per un numero negativo.
— Il C64 può stampare numeri interi o frazionari. Per i numeri frazionari si usa un . (convenzione anglosassone) invece della virgola degli italiani. Si usa il sistema decimale: non ci si deve preoccupare del binario. Segnaliamo pure che gli zero sono sbarrati (Ø) così da non essere confusi con la lettera O.
— Il C64 stampa al massimo 9 cifre significative. Ne usa un po' di più nella rappresentazione interna per fare i calcoli, ma non ne "mostra" più di 9. La rappresentazione interna è sempre frazionaria ma se, a parte l'imprecisione dei calcoli, il numero è abbastanza vicino ad un intero, verrà stampato come intero.

?14.99999999 dà 15

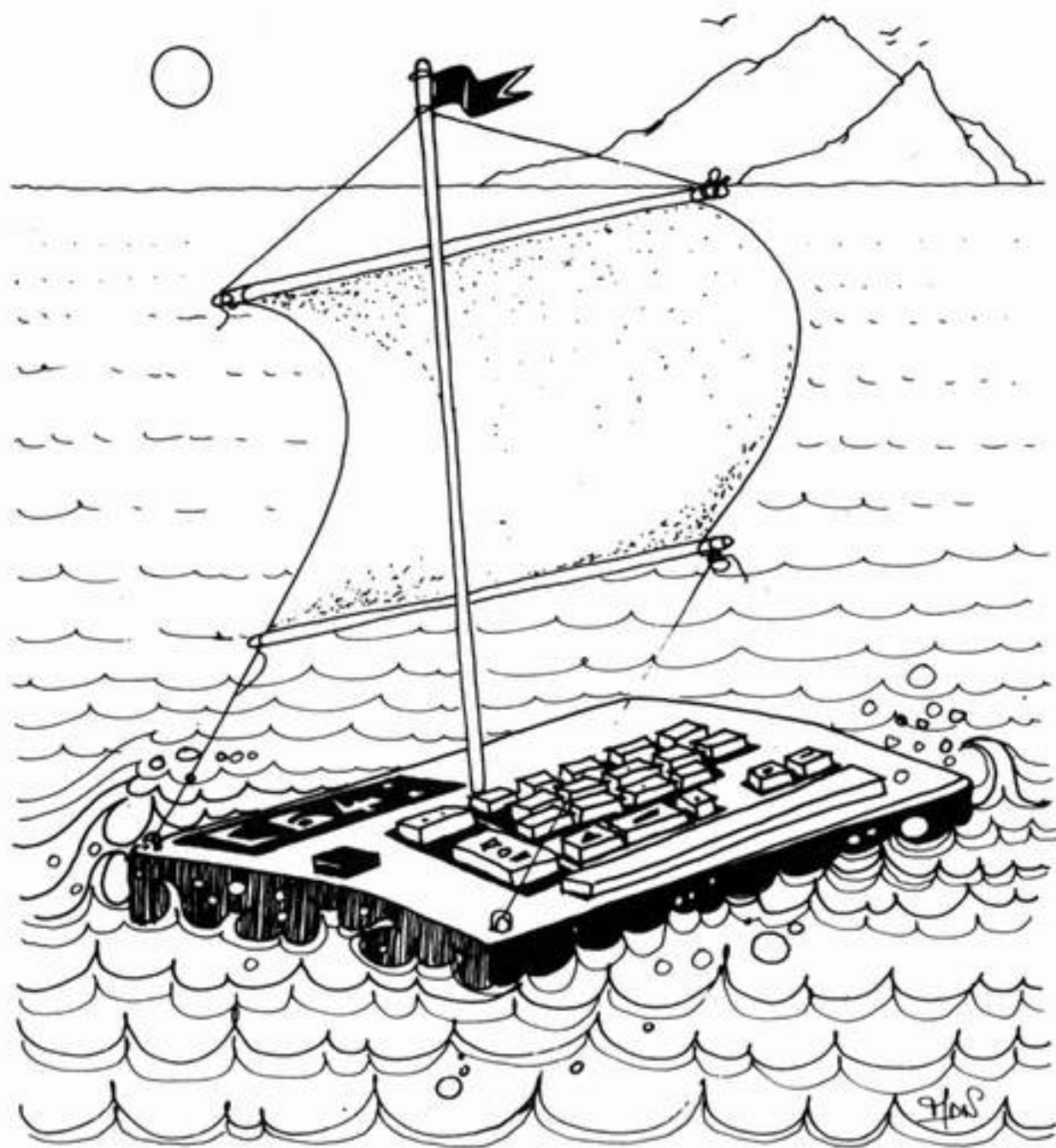
— Se il numero da stampare è minore di 0.1 o maggiore di 999999999, viene stampato in "notazione mobile", cioè nella forma:

Sx.xxxxxxxx Estt (x e t sono cifre, S è il segno del numero, s il segno dell'esponente).

? 0.000123 dà 1.23E—04
che si interpreta come $1{,}23 \times 10^{-4}$

?—1000000000 dà —1E09
che si interpreta come -1×10^{9}

Ecco conclusa la nostra presa di contatto con il C64. Speriamo non sia stata troppo ardua, e vi incoraggiamo a provare altri esempi, per familiarizzarvi bene.
Ora siamo pronti ad usare il C64 in modo programmato.

Capitolo 2

Istruzioni fondamentali

IL NOSTRO PRIMO PROGRAMMA

Pur cominciando con nozioni semplicissime passiamo ad una programmazione più elaborata in modo differito.
Ogni trattamento d'informazioni, quindi ogni programma, comprende tre tappe fondamentali, immutabili:

— l'*acquisizione* o immissione dei dati da trattare;
— il *trattamento* dei dati cioè il calcolo dei risultati;
— l'*uscita* dei risultati.

Avremo dunque in BASIC tre istruzioni fondamentali corrispondenti a questre tre azioni; le ritroviamo nel programma A.1 il cui scopo è semplicemente di calcolare la superficie di un cerchio di raggio R ($S = \pi R^2$).

Programma A.1

```
10 INPUT R
20 S=π*R↑2
30 PRINT S
```

I programmi vengono identificati da una lettera seguita da un numero di versione.
L'istruzione 10 corrisponde all'immissione dalla tastiera del raggio R. Quando avrete battuto RUN, in esecuzione di tale istruzione, il C64 visualizzerà sullo schermo un ? e il cursore lampeggiante. Si metterà in attesa che battiate un valore del raggio. Quando l'avrete fatto, senza dimenticare il 'Return', il C64 assegnerà quel valore alla variabile e passerà all'istruzione successiva.
L'istruzione 20 è l'istruzione di calcolo vero e proprio della superficie

alla quale abbiamo dato il nome di variabile S. L'espressione aritmetica BASIC è una copia quasi conforme (i segni di operazione sono obbligatori) della formula matematica che interviene. Notiamo che il BASIC del C64 conosce π, il che semplifica la battitura.
L'istruzione 30 chiede semplicemente la visualizzazione del risultato 7.06858347, come abbiamo visto nel modo diretto.
Tutto questo è semplice vero? Ebbene ciò racchiude il 90% di tutto il BASIC, dato che tutto si basa sulle tre operazioni fondamentali che abbiamo citato.
Benché sia semplice, il programma che abbiamo appena scritto funziona in modo soddisfacente: per calcolare la superficie di un cerchio, si batte RUN poi, appena il C64 visualizza un punto interrogativo, si batte il valore del raggio considerato, e appena premiamo il tasto 'Return', il risultato appare sullo schermo. Ed ogni volta che battiamo RUN, il programma viene rieseguito per un raggio nuovo: possiamo avere la superficie corrispondente a tanti raggi quanti ne vogliamo. Il fatto di eseguire un programma non lo "consuma", cioè esso rimane in memoria e, appena battiamo il comando diretto RUN seguito da 'Return', il programma presente in memoria si esegue.

Qualche perfezionamento

Tuttavia, il comportamento di questo programma ha alcuni inconvenienti ai quali rimedieremo; questo ci permetterà di vedere nuove istruzioni BASIC o nuove forme di quelle che conosciamo. Seguiremo del resto questo metodo per tutto il libro.

Obiezione n. 1: quando il C64 visualizza il ? nel corso dell'istruzione INPUT, nulla indica che è un raggio che si aspetta; ciò non ci disturba poiché lo sappiamo, ma una persona che non conoscesse il programma non saprebbe che rispondere al punto interrogativo. Perfino l'autore di un programma, nel caso di un programma che abbisogna di molti dati, può aver bisogno che gli si ricordi in che ordine deve inserirli.
Riassumendo, visualizzare un messaggio che dica all'utente quali sono i dati attesi dall'INPUT, sarebbe la cosa desiderabile. Gradiremmo, ad esempio, avere un messaggio del tipo:

RAGGIO DEL CERCHIO? o: DATE UN RAGGIO?

Ebbene BASIC lo permette molto facilmente. Infatti possiamo usare INPUT nella forma:

INPUT "testo"; variabile

A questo punto, il testo tra apici apparirà sullo schermo, seguito dal punto interrogativo abituale di INPUT, e basta rispondere come nel caso precedente.

Così, se sostituiamo l'istruzione 10 del programma A.1 con:

10 INPUT "RAGGIO DEL CERCHIO"; R

il nostro problema sarà totalmente risolto. Il punto e virgola che separa il testo dalla lista di variabili è imperativo, come pure gli apici che delimitano il testo.
Va bene, ma come sostituire la vecchia istruzione 10 con la nuova? Semplicemente battendo la nuova istruzione 10, senza dimenticare di iniziare con il numero. Essa andrà a sostituire quella vecchia nella memoria dove il programma è conservato.

Obiezione n. 2: quando è visualizzato il risultato, otteniamo un numero, ma nulla indica che si tratta della superficie del cerchio. Gradiremmo anche qui che un messaggio opportuno ci illuminasse. Nel caso di un programma che fornisse molti risultati, sarebbe del tutto indispensabile che ogni risultato fosse identificato.
Come quello precedente, questo problema si risolve facilmente in BASIC. Basta sapere che si può, mediante PRINT, visualizzare qualsiasi testo racchiuso tra apici. Così potremo sostituire l'istruzione 30 con:

30 PRINT "SUPERFICIE = ",S

e sarà detto tutto.
Ora avremo un dialogo della forma (sottolineiamo ciò che è battuto dall'utente):

```
RUN
RAGGIO DEL CERCHIO? 10
SUPERFICIE = 314.159265
READY.
```

Obiezione n. 3: il nostro programma è nettamente migliorato, soprattutto dal punto di vista del dialogo uomo-macchina, e ogni entrata-uscita deve presentarsi così. Ma rimane ancora un piccolo elemento di sconforto: quando si vogliono trattare più raggi, bisogna ogni volta battere RUN, il che è un po' fastidioso.
BASIC ha una risposta: aggiungeremo alla fine del programma un'istruzione che dice al C64 "ricomincia l'esecuzione dall'inizio (= l'istruzione 10)".
Aggiungeremo l'istruzione:

40 GOTO 10

Sapendo che "GOTO" significa "andare a", il funzionamento è evidente: ogni volta che è stato fatto un calcolo e il suo risultato è stato stampato, il C64 arriva all'istruzione 40 che lo rimanda in 10 dove si chiede un nuovo raggio e così via...
Disponiamo ora del programma A.2 che, benché sia stato facile da scrivere, ha un comportamento molto comodo. Esso fa apparire una nozione importante, la nozione di **ciclo**. Una struttura composta da

un gruppo di istruzioni che verranno eseguite più volte si chiama un ciclo. La possibilità di eseguire cicli, quindi di eseguire calcoli iterativi, è un punto forte dei computers.

Programma A.2

```
10 INPUT "RAGGIO DEL CERCHIO"; R
20  S = π * R↑2
30 PRINT "SUPERFICIE =",S
40 GOTO 10
```

N.B. Nelle liste che diamo come qui sopra, stampiamo una riga per istruzione, anche se questa appare su due righe sullo schermo del C64 a causa del suo limite di 40 colonne di larghezza dello schermo.

Uscita da INPUT per STOP-RESTORE

Il ciclo che abbiamo appena visto nel programma A.2 risolveva un problema ma ce ne pone un altro: infatti, non finisce mai. All'infinito il C64 chiederà un nuovo raggio, e un altro ancora...
In realtà è normalmente proibito inserire un tale ciclo infinito in un programma; deve essere sicuro che ogni programma termini dopo un tempo finito. Vedremo in seguito istruzioni che permettono di stabilire cicli dei quali si è certi che terminino. Tuttavia il male è minore quando il ciclo infinito contiene un'istruzione INPUT: in tal caso disponiamo di un modo per uscirne.
Infatti, ad ogni iterazione, quando esegue l'istruzione INPUT, il C64 s'interrompe nell'attesa del dato che dobbiamo fornire. In quel momento, se battiamo contemporaneamente i tasti 'Stop' e 'Restore', allora il C64 esce dal programma, svuota lo schermo e otteniamo il messaggio READY.
Il programma A.2 è dunque corretto: possiamo avere la superficie di tutti i cerchi che vogliamo, e quando non ne vogliamo più, battiamo i tasti 'Stop' e 'Restore' per terminare.

Stop e CONT

Sfortunatamente può accadere che la situazione si accomodi meno bene. Se, in seguito ad un errore di programmazione, il computer si perde in un ciclo senza fine e gira all'infinito senza fornire risultati, se ne può riprendere il controllo?
Sì! Basta premere il tasto 'Stop'. Si ottiene immediatamente il messaggio:

BREAK IN... (interruzione all'istruzione numero...)
READY

Si possono quindi far stampare alcune variabili per vedere se tutto è

corretto. Infatti può succedere che non si ottengono risultati per molto tempo, semplicemente perché i calcoli sono lunghi e non perché si ha un ciclo infinito. In quel caso, si può far riprendere l'esecuzione nel punto in cui si era fermata, battendo in modo diretto il comando: CONT.
Che differenza c'è tra 'Stop' da solo e 'Stop' con 'Restore'?
'Stop' da solo ferma l'esecuzione del programma in modo che possa riprendere con CONT.
Ma 'Stop' da solo è inoperoso quando il programma si trova in attesa su INPUT: non si può interrompere se non mediante 'Stop' - 'Restore'. 'Stop' - 'Restore' ha effetti diversi da quelli di una semplice istruzione: il programma non può più riprendere con CONT, lo schermo è svuotato e vedremo più tardi che se erano stati cambiati, i valori dello schermo e del quadro sono riportati ai loro valori standard (fondo azzurro, quadro e caratteri azzurro chiaro) che si hanno al momento dell'accensione.
'Stop' e 'Restore' creano quindi una specie di azzeramento della macchina, un po' come quando si spegne il C64 e poi lo si riaccende. Esiste tuttavia un'importante differenza: quando spegnete il C64 e lo riaccendete, anche il più presto che potete, tutto il contenuto della memoria viene cancellato, mentre con 'Stop' e 'Restore' il programma che sta in memoria viene conservato.

Complementi: senza pretendere di essere completi (il miglior modo di imparare tutte le particolarità di un linguaggio, è la pratica) c'è un paio di dettagli supplementari che dobbiamo darvi su INPUT e su PRINT.

Inserimento di più dati: possiamo inserire più dati in una stessa istruzione INPUT. Ad esempio, se, invece di calcolare la superficie del cerchio, volessimo calcolare il volume di un cilindro, bisognerebbe dare il raggio ma anche l'altezza H.
Si potrebbe usare un'istruzione della forma:

```
10 INPUT "RAGGIO, ALTEZZA", R, H
```

e, in risposta, si dovrebbero ora battere due numeri separati da una virgola:

```
RAGGIO, ALTEZZA? 15.5, 20 'Return'
```

Non confondete il punto di 15.5 che, in notazione inglese, separa parte intera e parte decimale di un numero, con la virgola che separa due numeri diversi che andranno in variabili diverse.

Piccola "posta del cuore": *in risposta ad un INPUT I, J nel quale dovevo inserire i valori 45 e 105, per errore ho piazzato male la virgola e battuto 4,5105. Che cosa succede?:* ...il C64 incomincerà i calcoli con i valori I = 4 e J = 5105, che non sono quelli che desiderate.

Altra domanda: *ad un INPUT I, J nel quale dovevo fornire due valori, ne ho dato uno solo, seguito da* 'Return'. *Che cosa succede?:* ...niente di grave, il C64 terrà conto di questo primo valore e, vedendo che gliene serve un altro, visualizzerà un doppio punto interrogativo. Quando abbiamo un INPUT a più variabili, possiamo raggruppare i dati che forniamo come vogliamo, poiché, finché non avrà avuto tutti i valori che aspettava, il C64 ve lo segnalerà visualizzando due punti interrogativi.
Esempio di dialogo: (istruzione 10 INPUT "I,J"; I,J)

I,J ? 45 'Return'
?? 105 'Return'

Al contrario, ora, batto troppi valori (esempio: 45, 105, 200 per INPUT I, J). Che cosa succede?: ...il C64 tiene conto degli n primi valori se ne aspetta n (nel nostro esempio I = 45 e J = 105) e stampa EXTRA IGNORED il che indica che i valori supplementari (e superflui) vengono ignorati.
E se non batto alcun valore, cioè, se in risposta ad un INPUT, faccio 'Return' subito?: ...Allora il C64 conserva i vecchi valori delle variabili che sono oggetto dell'INPUT: non dà doppio punto interrogativo.

Espressione aritmetica in un comando PRINT: finora abbiamo chiesto la stampa o di un titolo o del valore di una variabile. Si tratta di casi particolari della legge generale che permette di mettere come elemento da stampare qualsiasi espressione aritmetica: il C64 effettuerà il calcolo e visualizzerà il valore ottenuto.
Battete

?"5 HA PER QUADRATO", 5↑2

Otterrete

5 HA PER QUADRATO 25

caso particolare di espressione aritmetica, si può mettere una variabile, ma anche una costante. Si potrebbe scrivere:

?5; "HA PER QUADRATO", 5↑2

invece dell'esempio precedente.
Quando facciamo stampare un titolo, sfruttiamo il fatto seguente: ogni testo racchiuso fra apici costituisce una costante stringa di caratteri.

Separazione degli elementi in una lista di stampa: è beninteso che laddove diciamo "stampa" bisogna intendere "visualizzazione sullo schermo", ma con il C64 basta una sola istruzione che vedremo più tardi affinché tutte le informazioni chiamate in causa da tutti i comandi PRINT appaiano alla stampante se se ne dispone.
Finiamo anche con il punto interrogativo; sul C64, il ? è una comoda abbreviazione dell'istruzione PRINT.
D'altronde, nei vari esempi precedenti, quando abbiamo voluto visua-

lizzare più informazioni nello stesso comando PRINT, le abbiamo separate ora con una virgola, ora con un punto e virgola. Qual è la differenza? Quando due zone sono separate da un punto e virgola, saranno stampate congiunte sulla linea, mentre se sono separate da una virgola, la stampa della seconda zona comincerà alla prossima colonna dal numero multiplo di 10 + 1, libera (ciò si chiama tabulazione). Provate:

```
? A, B
0                    0
? A ; B
0    0
```

Perché non sono attaccati nel secondo esempio gli zeri? Perché ogni numero stampato comporta un segno (qui si ha uno spazio poiché i numeri sono positivi) ed è seguito da uno spazio.
Cosa avviene con le stringhe di caratteri?
Provate:

```
"BUON"; "GIORNO"
BUONGIORNO
?"BUON", "GIORNO"
BUON          GIORNO
?"EHI BUONGIORNO", "SIGNORE"
EHI BUONGIORNO          SIGNORE
```

Nell'ultimo esempio, siccome la O finale di BUONGIORNO stava in colonna 14, SIGNORE è mandato in colonna 21.
Ancora due indicazioni:

— Se non mettessimo alcun separatore, il C64 farebbe come se avessimo messo un punto e virgola. Ma ciò è possibile soltanto fra due stringhe di caratteri, o fra stringa e variabile.
— Si può terminare il comando PRINT con una, o con un; mentre non c'è nulla da separare. L'effetto di ciò è che il prossimo ordine PRINT scriverà sulla stessa linea, in modo concatenato o con una tabulazione, a seconda che avremo messo, o;
Provate i programmi:

1	2	3
10 ?"BUON"	10 ?"BUON";	10 ?"BUON",
20 ?"GIORNO"	20 ?"GIORNO"	20 ?"GIORNO"
BUON	BUONGIORNO	BUON GIORNO
GIORNO		

Tutto proviene dal fatto che un ordine PRINT normale effettua un ritorno carrello per terminare, che è soppresso dal ; e dalla ,. Invece, se vogliamo spostarci di una linea in avanti senza stampare nulla, basta mettere PRINT e basta.
Precisiamo infine che, se per un PRINT, la , o il ; non fanno grande

differenza tranne se vogliamo curare l'impaginazione, per INPUT sono essenziali e per niente intercambiabili: le variabili da immettere devono essere separate da virgole (così come i dati al momento in cui li forniamo) e, se c'è un messaggio di titolo, deve essere separato dal resto da un punto e virgola.

ESERCIZI

Benché abbiamo visto soltanto tre istruzioni BASIC, abbiamo già in mano possibilità immense, poiché esse coprono le tre operazioni fondamentali di ogni trattamento:

— immettere i dati;
— calcolare;
— far uscire i dati.

Verifichiamolo sui due esercizi che vi raccomandiamo fortemente di svolgere senza guardare la soluzione alla fine del volume.

Esercizio 2.1 *Modificare il programma A.2 per calcolare il volume di cilindri di raggio R e altezza H.*

Esercizio 2.2 *Scrivere un programma di struttura analoga, ma per calcolare l'interesse dato da un certo capitale piazzato ad un certo tasso per N anni (interessi composti annualmente).*

Nota: la numerazione degli esercizi è fatta sotto la forma capitolo. Numero.

Capitolo 3

Comandi fondamentali

Un programma inserito in modo diretto può essere usato soltanto in congiunzione con un certo numero di comandi in modo diretto, che dicano al sistema di gestione cosa vogliamo fare.
Abbiamo già visto il più fondamentale di questi comandi: RUN che permette di eseguire il programma.
Ma si possono fare altre operazioni su di un programma. Lo si può listare, cioè stampare le sue istruzioni, il che è utile, soprattutto per ricercare errori, o se si progetta una modifica. Lo si può correggere e, per questo, il C64 è estremamente comodo. Lo si può salvare su cassetta per poterlo usare ulteriormente senza doverlo ribattere.
Il C64 possiede tutto un ambiente di comandi che permettono queste operazioni. È indispensabile che le vediamo ora.

LIST

LIST da solo fornisce sullo schermo la copia di tutto il programma presente in memoria. Se non c'è un programma, ad esempio subito dopo l'accensione, il C64 visualizza READY immediatamente. Si dice che si ottiene la lista, o in italinglese il listing del programma.
Se il programma è molto lungo, e quindi non sta nelle 25 righe dello schermo, la lista scorrerà sullo schermo (una linea appare in basso quando una linea sparisce in alto) e la lettura sarà difficile. Per facilitarla, possiamo rallentare la velocità di scorrimento tenendo premuto il tasto 'CTRL'.
Potete provarci, ma i programmi che vi abbiamo scritto finora sono un po' corti perché si veda bene.
Due particolarità sono apparse sulle liste che abbiamo potuto ottenere facendo qualche prova:

— le istruzioni appaiono sulla lista nell'ordine dei numeri crescenti, cioè lo stesso ordine dell'esecuzione, quale che sia l'ordine nel quale sono state battute;
— se, per istruzioni di stampa, abbiamo usato il punto interrogativo, sulla lista è la parola PRINT che appare.

Supporremo nel seguito di avere in memoria il programma A.2 del capitolo precedente.

Liste parziali

Si può listare un'istruzione sola, dandone il numero: *LIST x*

```
LIST 20
20 S = π * R↑2
```

Per listare tutte le istruzioni comprese tra le istruzioni di numero x e y, si batte *LIST x-y*

```
LIST 20-30
20 S = π * R↑2
30 PRINT "SUPERFICIE =", S
```

Gli estremi — se esistono — sono compresi.
LIST 15-35 darebbe lo stesso risultato dell'esempio precedente.
LIST —x: lista dall'inizio fino alla linea x:

```
LIST —20
10 INPUT "RAGGIO DEL CERCHIO"; R
20 S = π * R↑2
```

LIST x—: lista dalla linea x fino alla fine:

```
LIST 30—
30 PRINT "SUPERFICIE =", S
40 GOTO 10
```

Come l'esecuzione di un programma, anche una lista può essere interrotta mediante il tasto 'Run/Stop'. In coppia con il LIST x—, ciò permette di listare un programma lungo per pezzi: si preme 'Stop' quando si vede che lo schermo si sta riempiendo.
Naturalmente, se non c'è nessuna istruzione nell'intervallo richiesto, riceviamo subito READY.

NEW

Quando battiamo un'istruzione BASIC in modo programmato (dunque con un numero) possono accadere due cose:

— o l'istruzione che abbiamo appena battuto ha lo stesso numero di un'istruzione già presente: allora viene a *sostituire* quella vecchia;
— oppure non esisteva un'istruzione dello stesso numero di quella che abbiamo appena battuto: allora — come potremo constatare chiedendo LIST — l'istruzione viene ad *intercalarsi* nel programma al posto implicato dal suo numero.

Ne risulta che, se vogliamo introdurre un programma completamente nuovo, e se le istruzioni nuove non corrispondono una per una a quelle del vecchio programma, rimarranno vecchie istruzioni in mezzo alle nuove che, naturalmente, perturberanno il funzionamento.
Il comando *NEW* ha lo scopo di eliminare questo inconveniente: il suo effetto è di sopprimere completamente il programma presente attualmente. È consigliato di usarlo prima di battere un nuovo programma.
NEW non deve essere confuso con un altro comando o istruzione *CLR* che ha, da parte sua, l'effetto di azzerare tutte le variabili. In pratica, NEW svuota la memoria programma mentre CLR svuota la memoria dati (l'una e l'altra sono due zone della stessa memoria).

Nota: In realtà, NEW contiene CLR, cioè, quando facciamo NEW, contemporaneamente tutte le variabili sono azzerate. Parimenti, RUN contiene CLR, di modo che all'inizio dell'esecuzione di un programma, tutte le variabili hanno valore 0 finché un'istruzione dà loro un altro valore.

RUN

Conosciamo già bene il comando RUN "tout court". Lo si può usare anche nella forma *RUN x* dove x è un numero d'istruzione. Ciò avrà l'effetto di lanciare l'esecuzione del programma, ma dalla riga x in poi.

Domanda: Ho un programma che comporta l'istruzione 10. Posso lanciarlo con RUN 10. Posso anche lanciarlo battendo GOTO 10 in modo diretto. Qual è la differenza?
— RUN 10 azzera le variabili, cosa che GOTO 10 non fa. Introducete il programma:

```
5 A = 3
10 ?A
```

e provate i dialoghi:

1	2	3
A = 5	A = 5	A = 5
RUN	RUN 10	GOTO 10

Effetto:

3 0 5

Nel primo caso, si passa sopra all'istruzione che dà ad A il valore 3. Nel secondo caso, RUN azzera A. Soltanto nel terzo caso l'effetto dell'assegnamento in modo diretto verrà conservato.

END

Il RUN numero permettere di costituire un programma a più parti tali che si esegua ora l'una ora l'altra. Basta battere RUN numero della prima istruzione della parte voluta.
Sì, ma supponiamo di aver eseguito la prima parte: ora andremo a finire nella seconda parte, cosa per niente desiderabile. Basta terminare ogni parte con un'istruzione *END* che vuol dire "tornare al livello di comando diretto". Dopo l'esecuzione di un'istruzione END, BASIC visualizza READY. Per buona regola, il nostro programma A.1 doveva terminare con un'istruzione:

40 END

Ma, in pratica, quando Basic arriva all'ultima istruzione di un programma senza incontrare END, si comporta come se tale istruzione ci fosse.

Scrittura di un programma

Dobbiamo ora vedere tutto un insieme di procedure che permettono di modificare o correggere un programma dovendo ribattere il meno possibile.
Da questo punto di vista, il C64 è particolarmente comodo. Ma prima di vedere queste procedure, dobbiamo fare una conoscenza più ampia con la tastiera. Avremmo potuto farlo in precedenza, prima ancora di procedere alle nostre prime prove, ma abbiamo potuto farne a meno, mentre ora è indispensabile.

La tastiera del C64

La tastiera del C64 è rappresentata nella figura 3.1. I quattro grossi tasti più chiari situati a destra sono ciò che chiamiamo tasti di funzioni pre-programmate: hanno un effetto particolare che è determinato da certi programmi, di giochi in particolare. Non ne parleremo più.
Possiamo considerare che ci sono tre tipi di tasti:

— i tasti ordinari, che quando premuti fanno apparire il carattere corrispondente sullo schermo: ad esempio, quando premete il tasto A, una A si stampa sullo schermo;

Fig. 3.1. La tastiera del C64.

— i tasti di modificazione di ciò che viene visualizzato ed i tasti speciali: come 'Return' o i tasti di movimento del cursore;
— i tasti di comandi che, premuti simultaneamente ad un secondo tasto, determinano la funzione del secondo tasto. I tasti di comando sono 'SHITF' (ne esistono due che sono perfettamente equivalenti), SHIFT LOCK (che blocca in posizione SHIFT), C= (si tratta dell'emblema del costruttore Commodore; lo designeremo spesso con il nome 'Commodore' o 'c=') e 'CTRL' (Controllo).

Ogni tasto ordinario ha, in generale, tre funzioni, rappresentate sull'alto e sul davanti del tasto.
I tasti "lettera" hanno la lettera sull'alto, e due simboli grafici sul davanti. Si ottiene la lettera battendo semplicemente il tasto. Si ottiene il grafico di destra premendo simultaneamente SHIFT e il tasto. Si ottiene il grafico di sinistra premendo simultaneamente 'Commodore' e il tasto.
Quando diciamo "premendo simultaneamente", ciò significa: premere prima sul tasto di comando (SHIFT, c=, CTRL) e mantenerlo schiacciato, premere sul tasto desiderato, lasciarlo, e, finalmente, lasciare il tasto di comando. Allenatevi alla vostra tastiera!
Su alcuni tasti sono scritti due caratteri (esempio: ?/). Ottenete il carattere di sotto premendo semplicemente il tasto (esempio: /). Ottenete il carattere di sopra (esempio: ?) premendo contemporaneamente SHIFT o 'Comodore'. Il tasto :/; ha lo stesso comportamento, π si ottiene con SHIFT o 'Commodore'. I tasti ← e 0 danno lo stesso carattere se si premono semplicemente, o con SHIFT, o con 'Commodore'.
I tasti numerici possono essere premuti con il tasto CTRL. Allora si ottiene la funzione indicata sul davanti. Descriveremo più avanti queste funzioni.

Tasti di movimento del cursore

Il cursore è il rettangolo lampeggiante che avete sullo schermo quando il C64 attende che battiate qualche cosa: segna la posizione sullo schermo dove apparirà il prossimo carattere che batterete.
I tasti di movimento del cursore spostano tale cursore senza stampa-

re caratteri né modificare caratteri già presenti sullo schermo, sui quali passa il cursore.
I tasti ⇒ e ⇓ creano, evidentemente, il movimento segnato. Il movimento di sopra si ottiene premendo contemporaneamente il tasto 'Shift'.

Esempio: ⇑ che raffigureremo nella forma a fa spostare di un gradino verso l'alto. Nello stesso modo, useremo le abbreviazioni seguenti per i simboli freccia in basso b, freccia a sinistra s, freccia a destra d.

Esercizio d'allenamento 3.1 *Effettuiamo il percorso dalla figura 3.2., prendendo cura di tornare al punto A.*

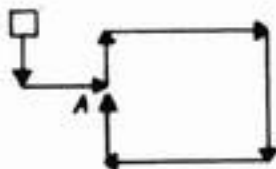



Fig. 3.2.

Il tasto 'Clr/Home' riporta il cursore alla sua posizione di origine in alto a sinistra dello schermo.
'Shift' 'Clr/Home' riporta il cursore in alto a sinistra, ma svuota anche la memoria di schermo, cioè sopprime ogni messaggio. Questo non va confuso con CLR o NEW: soltanto lo schermo è svuotato, il programma e i dati rimangono.
Il tasto 'Inst/Del' sopprime il carattere immediatamente alla sua sinistra, permettendo così un'immediata correzione degli errori di battitura.
Vogliamo battere BUONGIORNO, ma ci accorgiamo di aver battuto BUONKI.
Battiamo una prima volta 'Del' : BUONK poi una seconda: BUON e non ci rimane che da continuare GIORNO.
Con 'Shift' è la funzione di inserzione quella che otteniamo.
Vogliamo battere BUONGIORNO, ma ci accorgiamo di aver battuto BUONIORNO.
Portiamo il cursore sulla I facendo quattro volte 'Shift' s : 0. Facciamo 'Shift' 'Inst/Del' il che apre uno spazio alla sinistra della posizione attuale del cursore. Il cursore viene sull'ultimo spazio aperto in tal modo: BUON IORNO.
Non ci rimane che da battere la G: BUONGIORNO.

Esercizio di allenamento 3.2 *Volevate battere BUONGIORNO e avete battuto BUONORNO.*
Questi ultimi tasti vi serviranno particolarmente per la correzione dei programmi.

Tasti di controllo del colore

Queste funzioni si ottengono premendo simultaneamente 'CTRL' e un tasto numerico. Il loro effetto è che tutti i caratteri battuti dopo l'invocazione della funzione saranno del colore corrispondete, e ciò fino a che non si chiederà un altro colore.

Esempio: Se battete 'Ctrl' RED BUONGIORNO 'Ctrl' GRN SIGNORA avrete sullo schermo BUONGIORNO in rosso e SIGNORA in verde.
La corrispondenza tra nomi inglesi e colori è la seguente:

Cifra	*Abreviazione*	*Termine inglese*	*Colore ottenuto*
1	BLK	black	nero
2	WHT	white	nianco
3	RED	red	rosso
4	CYN	cyan	turchese
5	PUR	purple	porpora
6	GRN	green	verde
7	BLU	blue	azzurro
8	YEL	yellow	giallo

NB. Se domandate Ctrl BLU, non vedrete quello che stamperete perché i caratteri verranno in azzurro su fondo azzurro. Ma, vedremo più tardi che si può cambiare il colore del fondo dello schermo: allora, i caratteri azzurri si vedrebbero.

I tasti 9 e 0 non agiscono sul colore. Essi producono ciò che si chiama il contrasto invertito ("reverse"): dopo 'Ctrl' 'RVS ON', tutto quello che batterete apparirà in bianco (colore di fondo generale dello schermo) su fondo colorito. Si ritorna al contrasto normale battendo Ctrl RVS OFF.

Esempio: Battete BUONGIORNO Ctrl RVS ON CARA Ctrl RVS OFF SIGNORA. Con 'c=' i tasti numerici danno altri colori riassunti nel capitolo VI. In particolare, 'c=' 7 dà l'azzurro chiaro dei caratteri all'accensione.

Altri tasti speciali

Abbiamo già visto i ruoli di 'Return' (ritorno-carrello e fine di messaggio), 'sbarra di spazio', 'Run/Stop' (di cui abbiamo visto soltanto la funzione Stop).
La pressione simultanea di 'Stop' e 'Restore' rimette il C64 nel suo stato all'accensione, in modo particolare dal punto di vista dei colori, e svuota lo schermo.
La pressione di 'Ctrl' da solo rallenta lo scorrimento sullo schermo quando c'è una stampa rapida.

Modo minuscole

All'accensione, i tasti lettere danno le lettere maiuscole, il grafico di destra (con Shift) e il grafico di sinistra (con 'Commodore').
Ebbene, esiste un modo nel quale si ottiene la lettera minuscola (con il tasto da solo), la lettera maiuscola (con Shift) e il grafico di sinistra (con 'Commodore'). Abbiamo dunque un comportamento per macchina per scrivere, con possibilità grafiche più limitate (ma rimangono tutti i grafici di sinistra).
Passiamo da un modo all'altro, e inversamente, premendo simultaneamente SHIFT e 'Commodore'. La trasformazione concerne tutto ciò che appare sullo schermo. I grafici di destra e le maiuscole non possono apparire simultaneamente.

Domanda: Come avere, sullo schermo, un rettangolo della taglia di un carattere completamente colorato?
Battete

'Ctrl' 'Rvs' 'spazio'

Per avere la bandiera italiana fate:

'Ctrl' 'Rvs' 'Ctrl' 'GRN' 'spazio' 'Ctrl' 'WHT' 'spazio' 'Ctrl' 'RED' 'spazio'.

Ripetizione

I tasti cursore, Inst/Del, e sbarra di spazio hanno la ripetizione automatica, cioè, se li manteniamo premuti, la loro funzione si ripete. È specialmente pratico per far viaggiare velocemente il cursore.

Modifiche e correzioni di un programma

Quando si è trovato un errore in un programma, o quando si vuole semplicemente apportargli una modifica, è importante poterlo fare comodamente, cioè dovendo ribattere meno informazioni possibile.

Sappiamo già:

Aggiungere o inserire una nuova istruzione: basta battere l'istruzione attribuendole un numero compreso tra quelli delle istruzioni tra le quali vogliamo intercalarla.
Da ciò risulta immediatamente un consiglio: quando scriviamo la prima versione di un programma, non bisogna dare numeri consecutivi, per poter operare inserzioni in seguito. Si suggerisce, ad esempio, di numerare di dieci in dieci.

Trasformare completamente un'istruzione: battiamo la nuova versione con lo stesso numero della vecchia. Appena battiamo 'Return' la sostituzione ha luogo.

Vediamo ora nuove possibilità:

Soppressione completa di una istruzione: basta battere il numero seguito da 'Return'.
Ne faremo una prova che ci servirà per il seguito. Supponiamo di avere il programma A.1 in memoria (altrimenti ribattetelo: è fastidioso, ma vedremo presto come ricuperare un programma senza ribatterlo, grazie alle cassette).
Facciamo un LIST, ma prima, battiamo 'Shift' 'Clr/Home' per avere il listing in alto sullo schermo.
Poi, battiamo 30 'Return' per sopprimere l'ultima istruzione. Verifichiamo con LIST che è stato fatto effettivamente. Non abbiamo più l'istruzione 30.
Per recuperarla, se a seguito di un errore non fosse quella che bisognava sopprimere, dobbiamo teoricamente ribatterla. Ebbene no! Non fintantoché l'istruzione è visualizzata. Infatti, se l'istruzione è visualizzata, sta nella memoria di schermo. C'è un modo per trasferirla nella memoria programma: è battere 'Return' mentre il cursore sta ovunque sulla linea dell'istruzione.
Proviamolo: Portiamo il cursore sulla linea 30 del primo listing; abbiamo sullo schermo:

▒ 30 PRINT S

Allora, battiamo 'Return'. Sembra non accadere nulla. Facciamo scendere il cursore verso il fondo dello schermo e facciamo LIST: vediamo allora che l'istruzione 30 è tornata nel programma.
Riassumendo, appena si batte 'Return' di fronte ad un tasto che inizi con un numero d'istruzione, esso diventa un'istruzione nella memoria programma. Questo ci servirà per tutte le altre procedure di modifica. È già quello che usiamo per sopprimere un'istruzione: creiamo una nuova versione vuota dell'istruzione, e Basic non tiene un'istruzione vuota.

Modifica di qualche carattere su di una linea

Qui si manifesta tutta la potenza del sistema di editoria del C64. Da quello che abbiamo appena visto, risulta che la procedura da osservare è la seguente:

1. Listare la linea da modificare (se non sta già sullo schermo).
2. Portare il cursore sulla linea da modificare e sui caratteri da cambiare.
3. Per ogni carattere da sostituire, battere sul posto il nuovo carattere. Se ci sono caratteri da sopprimere, battere 'Inst/Del', se ce ne sono da inserire, usare 'Shift' 'Inst/Del'.
4. Quando la linea è nello stato desiderato, fare 'Return'.

Si può modificare una linea in tante tappe quante si vuole: basta tornare sulla linea per farci altre modifiche dopo aver battuto 'Return'.

Modifica del numero di una linea: si tratta di un caso particolare del precedente: ora è il numero di linea che cambiamo, ma attenzione! Una volta battuto 'Return' ci sono due linee identiche, una al vecchio numero e una al nuovo. Questo può essere particolarmente prezioso per risparmiare il tempo di battitura quando si ha tutta una serie di linee molto simili da battere.
Supponiamo si voglia avere (è un caso didattico):

```
50 GOTO 10
70 GOTO 10
90 GOTO 10
95 GOTO 12
```

Si batterà:

50 GOTO 10 'Return'

poi, cursore sul 5:

7 'Return'

poi, cursore sul 7:

9 'Return'

poi, cursore sullo 0:

5 (cursore sullo 0) 2 'Return'

Ogni volta avremo sullo schermo l'ultima riga battuta, ma facendo LIST ci accorgiamo di possedere tutte le linee desiderate.

Esercizio 3.3 *Modificare il programma A.1 affinché calcoli, non la superficie del cerchio, ma il volume della sfera di raggio R.*

Diamo l'esercizio non tanto per il calcolo quanto per allenarsi ad effettuare la modifica. Si vede facilmente che basta cambiare il nome del risultato in V (sarebbe possibile mantenere S, ma vogliamo identificatori eloquenti). Quindi 30 deve diventare 30 PRINT V, mentre 20 deve diventare 20 V = 4/3*π*R↑3.
La modifica di 30 è ovvia, portiamo il cursore sulla S, e battiamo V 'Return'.
Per verificare l'importanza di 'Return', battete la V, portate il cursore in un luogo vuoto dello schermo e fate 'Return' solo allora. Facendo LIST, vedrete che la modifica non è stata eseguita.
Per l'istruzione 20, portiamo il cursore sulla S che cambiamo in V, poi, col cursore su π, facciamo quattro volte 'Shift' 'Inst/Del' battiamo il 4/3* negli spazi così liberati, poi, col cursore sul 2 battiamo 3.

Domanda: non manca qualcosa?: Sì! il 'Return'.
Naturalmente sarebbe stato meglio modificare il programma A.2; è l'argomento dell'esercizio 3.4.

Esercizio 3.4 *Fare la stessa modifica sul programma A.2.*

Tutto dovrebbe andare bene, ma la modifica di testi fra apici (SUPERFICIE deve pur diventare VOLUME) può creare problemi sui quali torneremo in seguito.

Esercizio di allenamento 3.5 *Fare il passaggio dal programma A.2 al calcolo del volume del cilindro* (Esercizio 2.1, soluzione in appendice).

Prima di vedere i comandi che riguardano le cassette, passiamo ad un altro esercizio che ci permetterà di affrontare un altro tipo di problema.

Esercizio 3.6 *Scrivere un programma analogo al programma A.2, ma che chieda, questa volta, la superficie di un cerchio e ne deduca il raggio.*

Il problema è che, questa volta, non abbiamo una formula nota da applicare, "la pappa non è fatta". Bisogna cercarla, quantunque non sia troppo difficile, per questo esercizio.
Tuttavia, e così sarà per tutti i problemi tranne quelli totalmente ovvi, il computer non è capace di trovare da solo la soluzione di un problema. Bisogna dargliela sotto forma di una successione ordinata di operazioni da eseguire. Una tale successione, che non è altro che una ricetta, si chiama, presso gli informatici sapienti, un algoritmo.
È quasi sempre più difficile trovare l'algoritmo che risolva un problema che non programmare questo algoritmo una volta trovato.
Torniamo al nostro esercizio. Si trova facilmente che la formula da applicare è $R = \sqrt{S/\pi}$. Come tradurremmo la radica quadrata? Ebbene, abbiamo la scelta, come spesso avviene in programmazione, tra ↑.5 (potenza 1/2), o richiamare la funzione matematica SQR (radice quadrata) che è una di quelle di cui disponiamo in Basic. Donde due soluzioni per l'istruzione 20 del programma A.3.

Programma A.3

```
10 INPUT "SUPERFICIE DEL CERCHIO"; S
20 R = SQR (S/π)
30 PRINT "RAGGIO =", R
40 GOTO 10
```

Altra formula di 20:

```
20 R = (S/π) ↑.5
```

NB. Nei vari listati di programmi che forniamo, quello che scriviamo su di una linea corrisponde ad una linea logica che può, sul vostro schermo, apparire su due linee a causa della limitazione dello schermo a 40 caratteri per linea.

Archiviazione di un programma su cassetta

(Se avete a disposizione un magnetofono, cosa che vivamente vi raccomandiamo).
Eccovi ora dei comandi particolarmente utili. Infatti, finora abbiamo scritto soltanto programmi cortissimi e ne abbiamo scritti pochi. Tuttavia, anche così, è fastidioso batterli più volte, e crea dei rischi di errori. Ora, se facciamo NEW, o se spegnamo il C64 semplicemente per andare a dormire, il programma è perso (la memoria viva RAM perde le proprie informazioni quando non è più alimentata). Fortunatamente, ci sono le cassette sulle quali possiamo salvare programmi e rileggerli in seguito.

Salvataggio

Munitevi di una cassetta vergine riavvolta (altrimenti riavvolgetela con il tasto REW del magnetofono). Notiamo, a questo proposito, che è raccomandato usare cassette corte (dati i tempi di lettura, conviene riporre soltanto pochi programmi su ogni cassetta) e di buona qualità.
Supponiamo di avere in memoria un programma prezioso, ad esempio il programma A.3 che abbiamo appena fatto. Piazzate la cassetta nel magnetofono del C64 e battete:

SAVE "RAGGIO-CERCHIO"

Il C64 risponde:

PRESS RECORD & PLAY ON TAPE

che vuol dire "premete sui tasti RECORD e PLAY (simultaneamente) sul magnetofono". Fatelo. Il C64 svuota lo schermo. Al termine, visualizza:

OK
WRITING RAGGIO-CERCHIO

Quando riappaiono READY ed il cursore, è terminato: il programma è stato scritto sulla cassetta col nome che abbiamo dato. Dobbiamo fare due osservazioni:

1. Il nome è facoltativo, ma è essenziale se vogliamo distinguere vari programmi su di una stessa cassetta.
2. Quando avrete premuto i tasti del magnetofono che vi indicava, il C64 ha risposto OK. Quindi, si è accorto che avevate premuto. Ma non distingue bene i tasti, dunque bisogna prendere cura di premere tutti i tasti voluti, soprattutto RECORD. Potremmo anche non mettere alcuna cassetta, il C64 crederebbe che l'archiviazione stia avendo luogo!

Verifica

Una volta scritto il programma su cassetta, e possiamo farlo più volte: basta ricominciare il comando SAVE, senza riavvolgere né liberare i tasti del magnetofono; in tal caso, il C64 non chiede di premere i tasti, ma mette in moto il magnetofono e si mette a scrivere, bisogna verificare il buon esito dell'operazione.
A questo scopo, riavvolgete la cassetta, e battete:

VERIFY "RAGGIO-CERCHIO"

Il C64 risponde:

PRESS PLAY ON TAPE

(siccome vogliamo leggere, non bisogna premere il tasto RECORD questa volta). Dopo che il tasto PLAY è stato premuto, il C64 visualizza:

OK
SEARCHING RAGGIO-CERCHIO (sto cercando)

poi, per ogni programma che troverà, visualizzerà:

FOUND nome (sto verificando)

Normalmente, dopo poco tempo, dovremmo ottenere:

OK
READY

Cos'è successo? Semplicemente il C64 ha letto il programma sulla cassetta e l'ha paragonato con il programma che aveva ancora in memoria. Se questi due sono identici, il salvataggio ha avuto buon esito, e stampa OK.
Altrimenti viene visualizzato il messaggio ?VERIFY ERROR. Allora bisogna riavvolgere e riprovare la verifica. Se l'errore si ripete, bisogna ricominciare il salvataggio; eventualmente su di un'altra cassetta (il programma non è perso, ecco l'interesse dell'operazione VERIFY). Se l'errore è persistente, bisogna rivolgersi al servizio di assistenza.

Caricamento

Per rimettere in memoria un programma precedentemente registrato su cassetta, usiamo il comando LOAD "RAGGIO-CERCHIO".
Il C64 risponde, come per VERIFY:

PRESS PLAY...
poi SEARCHING...
poi FOUND... (per ogni programma trovato)
poi LOADING (quando ha trovato il programma ricercato)
finalmente READY.

Possiamo quindi listare il programma o eseguirlo mediante RUN.
Se otteniamo il messaggio diagnostico ?LOAD ERROR, dobbiamo riavvolgere e riprovare, poi rifare la prova su di un'altra cassetta sulla quale avremo precauzionalmente eseguito un secondo salvataggio. In caso di fallimento: servizio di assistenza (spesso si tratta di effettuare una regolazione semplicissima).

NB. I messaggi citati appaiono soltanto alla fine poiché, durante il funzionamento della cassetta, il C64 svuota lo schermo.

Nomi abbreviati

Possiamo specificare un nome abbreviato rispetto a quello specificato al momento del SAVE. Ad esempio, avremmo potuto scrivere:

LOAD "RA"

Il C64 caricherà il primo programma incontrato sul nastro dal nome compatibile con l'abbreviazione. Esempio, se prima di RAGGIO-CERCHIO c'è un programma RADIO sul nastro, verrà caricato RADIO.
È anche possibile non specificare alcun nome e scrivere LOAD e basta. In tal caso, viene caricato il primo programma incontrato sulla cassetta.

Caricamento ed esecuzione riuniti

Se si premono simultaneamente i tasti 'Shift' e 'Run/Stop', si ottiene l'equivalente della sequenza di comandi LOAD e RUN, cioè viene caricato il primo programma incontrato sulla cassetta e viene subito lanciata la sua esecuzione.

RICAPITOLAZIONE

Ormai abbiamo visto le più fondamentali istruzioni dei programmi:

aritmetico
INPUT-PRINT
GOTO-END

Abbiamo visto i comandi più utili:

RUN e LIST
SAVE, VERIFY e LOAD

così come le procedure di correzione dei programmi.
Se l'enunciato di alcune delle parole-chiave precedenti non risveglia in voi alcun ricordo, vi consigliamo, prima di proseguire, di rileggere le pagine che le riguardano.

Ora siamo pronti per affrontare la seconda serie di programmi che ci permetterà — sempre per varianti successive — di accrescere il nostro "arsenale" di istruzioni Basic.

Capitolo 4

Basi della programmazione

Se non avete nulla in contrario, faremo un gioco. I programmi di giochi formano una categoria importante dei programmi per piccoli sistemi. Non sono tutti folli: alcuni sono molto elaborati e spesso divertentissimi. Inoltre, la presentazione sotto forma di gioco di certi programmi pedagogici li rende più attraenti e quindi più efficaci. Desideriamo che, al termine della lettura di questo libro, siate capaci, anche voi, di scriverne.
Per il momento, il nostro gioco sarà un po' semplice in partenza, ma migliorerà progressivamente. Si tratta del gioco "indovina un numero". Il programma conosce un numero (fisso) e legge il tentativo del giocatore; se il giocatore ha indovinato, stampa "vinto", altrimenti stampa "perso".

L'istruzione *IF*

Per realizzare quanto richiesto, di che cosa abbiamo bisogno? Di una istruzione capace di testare una condizione (che qui sarà numero dato dal giocatore = numero nascosto), cioè di valutare se è vera o falsa; se è vera, di andare ad un certo punto del programma (in questo caso stampare "vinto") altrimenti di andare ad un altro punto del programma (in questo caso, stampare "perso").
Ebbene, questa istruzione esiste, è l'istruzione IF. La sua forma principale è:

```
n  IF condizione THEN istruzione
n' IF...
```

Il comportamento è questo: se la condizione è vera, si effettua l'istruzione che segue THEN, poi l'istruzione della linea successiva n'; se la condizione è falsa, si passa direttamente all'istruzione della linea n'.

Un caso particolare si presenta quando l'istruzione che segue THEN è un GOTO:

```
n IF condizione THEN GOTO n"
n'
...
n"
```

Allora, se la condizione è vera, si va in n"; se la condizione è falsa, si va in n'.
In pratica basta scrivere una delle due parole THEN e GOTO: Esempio:

```
10 IF A=B THEN 50
20 IF A+B<C GOTO 100
```

Siamo pronti per scrivere la nostra prima versione del programma "indovina un numero".

PROGRAMMA B.1

```
20 INPUT"INDOVINA UN NUMERO";A
30 IF A=3.25 GOTO 60
40 PRINT"PERSO!"
50 END
60 PRINT"VINTO"
```

Le istruzioni INPUT e PRINT ci sono già familiari. L'istruzione END fa terminare il programma una volta scritto PERSO! Non ce n'è bisogno dopo 60 poiché è l'ultima istruzione del programma.
L'istruzione IF che usiamo è del terzo tipo: IF... GOTO. Il numero da indovinare è 3.25; il numero proposto alla tastiera è A; la condizione da testare è "A è uguale a 3.25?" Ebbene, si scrive A = 3.25. È semplice.
Dal punto di vista del funzionamento del gioco, sono tante le obiezioni da formulare sul programma B.1. Lo faremo presto, e ci sarà di aiuto per scoprire nuove istruzioni Basic. Ma la forma attuale ci ha permesso di usare l'istruzione IF, che è una delle più importanti del Basic, e dobbiamo ancora vedere alcune cose a proposito di questa istruzione, in particolare, le diverse istruzioni che possono seguire l'IF.
La prima forma di condizione è: espressione aritmetica relazione espressione aritmetica, come 2*A+4 < B↑3.
Ogni espressione aritmetica viene valutata prima di esaminare la relazione. Gli operatori di relazione che si possono usare sono:

= uguale	< > diverso
< minore	< = minore o uguale
> maggiore	> = maggiore o uguale

"diverso" si scrive "minore o maggiore", abbastanza logicamente.

La seconda forma di condizione è una combinazione di relazioni della prima forma mediante gli operatori logici AND (e), OR (o) e NOT (non).

C1 AND C2	è vera solo se le condizioni C1 e C2 sono entrambe vere.
C1 OR C2	è vera non appena è vera una almeno delle condizioni C1 e C2.
NON C	è vera se C è falsa, e falsa se C è vera.

— Andare in 100 se contemporaneamente C è maggiore di 2∗A+4 e B è minore di 3:

IF C > 2∗A + 4 AND B < 3 GOTO 100

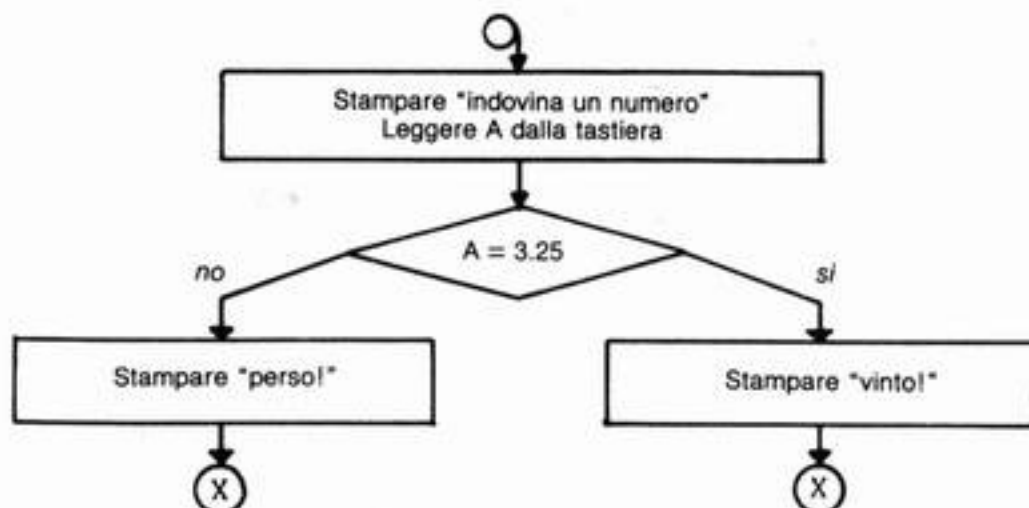
— Stampare SI se X è fuori dall'intervallo [1, 2[(cioè X < 1 o X > = 2):

IF X < 1 OR X > = 2 THEN PRINT "SI"

Esercizio 4.1 *Riprendete il programma A.3. Provate a fornire una superficie negativa.*
Si ottiene il messaggio:

?ILLEGAL QUANTITY ERROR IN 20

cosa normale, dato che si cerca di calcolare la radice quadrata di un numero negativo: non esiste un cerchio di superfice negativa.
Un programma scritto bene deve garantirsi contro tali errori dell'operatore: ad esempio, un programma di scacchi deve verificare che il colpo proposto dal giocatore sia legale.
Lo scopo dell'esercizio è di immettere una tale garanzia nel programma A.3. Aggiungete al programma un'istruzione che rimandi in 10 per chiedere un'altra superficie fintantoché la superficie fornita non è positiva. Si può pure, inoltre, stampare un messaggio di protesta.

Adesso abbiamo visto la nostra prima istruzione veramente elaborata. Infatti, rende capace il C64 di prendere decisioni in funzione delle varie situazioni che possono risultare dai dati. In realtà, le decisioni le prendete voi quando preparate il programma, e se per caso dimenticate un caso possibile, il programma si comporterà scorrettamente se il caso si realizzerà.
Di decisione in decisione, il cammino può ramificarsi in modo complesso.
Gli *schemi a blocchi*, detti anche *organigrammi*, possono diventare necessari per raccapezzarsi.
Prima di passare al miglioramento del nostro programma di gioco, studieremo lo schema a blocchi del programma B.1.

È composto di blocchi, che specificano le varie operazioni, collegati tra loro da frecce che rappresentano l'ordine di successione delle operazioni. La forma stessa del blocco indica al primo sguardo la natura dell'operazione.
Tra le forme del blocco, si distinguono essenzialmente:

— il *rettangolo*, che ha una sola entrata ed una sola uscita e rappresenta qualsiasi operazione imperativa;
— il *rombo*, che ha una sola entrata, ma due o più uscite, e rappresenta le operazioni di test.

Si usano i segni speciali ⍜ e ⊗ per segnare l'inizio e la fine del trattamento.
È sempre raccomandabile tracciare lo schema a blocchi prima di intraprendere la redazione del programma: non è mai una perdita di tempo.

Esercizio 4.2 *Tracciare lo schema a blocchi del programma A.2. Tracciare lo schema a blocchi del programma dell'esercizio 4.1.*

Esercizio 4.3 *A quali parole-chiave Basic viste finora corrispondono i blocchi o segni:*

Perfezionamento del programma B

Bisogna pur ammettere che, nella sua prima versione, il gioco non è particolarmente interessante. Ma siamo ora in grado di migliorarlo. La prima obiezione non sarà risolta se non proprio alla fine. È un peccato perché riguarda la credibilità stessa del gioco.

Infatti, basta battere LIST per avere la lista del programma, e venire così a conoscenza del numero da indovinare. Supporremo per un po' che il giocatore non conosca il comando LIST, oppure che stia al gioco.
Dobbiamo notare che questa obiezione si presenta pure, anche in programmi più elaborati: ad esempio, se giocate alla battaglia navale col computer e se conoscete bene il programma, potete far stampare le variabili che contengono le coordinate delle navi avversarie.
Supponendo ora che giochiate senza barare, obietterete che il gioco è difficilissimo, addirittura non equo: avete poche possibilità di trovare il numero al primo colpo, come esige il programma. È chiaro che bisogna lasciare al giocatore più possibilità.
Lo si può fare molto semplicemente: basta sostituire l'istruzione 50 del programma B.1 con 50 GOTO 10 ed ora abbiamo diritto ad un numero illimitato di tentativi.
Malgrado ciò, il gioco rimane assai aleatorio per due motivi:

— in caso di errore, nulla dice al giocatore se è lontano o vicino dal risultato;
— il giocatore deve azzeccare proprio il valore giusto, cioè deve fornire il numero con la precisione con la quale lavora il C64, cioè 10^{-9}.

Risolveremo il primo problema calcolando e stampando ogni volta la percentuale di errore E uguale al numero proposto meno il numero da trovare diviso il numero da trovare, il tutto moltiplicato per cento.
La percentuale di errore verrà stampata in valore assoluto, dunque senza indicare il senso dell'errore, per lasciare al gioco una certa difficoltà.
Per quanto riguarda l'errore possibile dovuto alla mancanza di precisione del personal non si confronterà il numero proposto con il numero da trovare, ma si considererà la risposta esatta se E, percentuale d'errore, sarà minore di 0,5%.
Il problema di precisione si pone ogniqualvolta si hanno calcoli da effettuare. Per risolverlo, basta sostituire un test di eguaglianza pura del tipo IF A = B con un test sul valore della differenza tra i due numeri del tipo A-B < soglia, dove la soglia è l'ordine di grandezza della precisione — o piuttosto dell'imprecisione — del C64. In realtà è il valore assoluto della differenza che si dovrebbe testare. Il nostro programma diventa:

PROGRAMMA B.2

```
20 INPUT"INDOVINA UN NUMERO";A
25 E=100*ABS(A-3.25)/3.25
30 IF E<0.5 GOTO 60
40 PRINT"ERRORE";E;"%"
50 GOTO 20
60 PRINT"VINTO!"
```

Nella linea 25, usiamo la funzione ABS (valore assoluto) che fa parte della biblioteca matematica del Basic del C64 la cui lista completa è data in appendice.
Se facciamo girare il programma così com'è, otteniamo, ad esempio, il messaggio:

ERRORE 10.7692308%

È ben certo che non sappiamo che farcene di tante cifre decimali. Come toglierle? Useremo un'altra funzione della biblioteca, la funzione INT, che prende la parte intera dell'argomento messo tra parentesi. Sostituiamo, nell'istruzione 40, E con INT(E). Questa volta, otteniamo una percentuale intera. Ma è troppo poco. Come fare per ottenere, mettiamo, due cifre decimali?
A questo scopo, moltiplichiamo E per 100 per far passare le due cifre decimali che vogliamo tenere nella parte intera, prendiamo lo INT di questo prodotto, che fa sparire le altre cifre decimali, poi, ridividiamo per 100:

INT (E * 100)/100

Tale è l'espressione che viene a sostituire E nell'istruzione 40 del programma B.2, ed abbiamo ora un messaggio soddisfacente.

Esercizio 4.4 *Vogliamo stampare il numero X con D cifre decimali. Scrivere l'istruzione corrispondente in modo diretto.*

Per costituire la versione 3 del nostro programma, vogliamo aggiungere un altro perfezionamento. Sebbene gradevole, quando il numero da indovinare è stato trovato, stampare il numero di tentativi che sono stati necessari. Si tratta, in pratica, del punteggio del gioco. Per questo introduciamo una nuova variabile N, alla quale sommiamo 1 ogni volta che un tentativo è stato fatto senza successo, e che stampiamo alla fine; ed ecco il programma B.3:

PROGRAMMA B.3

```
20 INPUT"INDOVINA UN NUMERO";A
25 E=100*ABS(A-3.25)/3.25
30 IF E<0.5 GOTO 60
35 N=N+1
40 PRINT"ERRORE";INT(E*100)/100;"%"
50 GOTO 20
60 PRINT"VINTO IN";N+1;"TENTATIVI"
```

L'istruzione 35 può sembrare paradossale, ma non dimentichiamo che il segno = non ha in Basic lo stesso significato che in matematica. In Basic, vuol dire: calcolare l'espressione che sta a destra, quindi

N + 1, e mettere il risultato nella variabile che sta a sinistra. Qui, è la stessa variabile, cosa perfettamente lecita, e questo è effettivamente come incrementare N.

Domanda: *La primissima volta che si incontra l'istruzione 35, si deve calcolare l'espressione N + 1; quale valore si prende per N?*

Fate bene a porre questa domanda che solleva il problema dell'*inizializzazione delle variabili*. Ma sappiamo che quando facciamo RUN, Basic azzera tutte le variabili, quindi anche N. Ora, quando cominciamo, abbiamo fatto 0 tentativi, dunque il valore iniziale automatico di N è adeguato. Se, in un altro problema, avessimo avuto bisogno di un altro valore iniziale che 0, allora si sarebbe dovuto fornirlo esplicitamente nelle prime istruzioni del programma; questo è un punto fondamentale: parecchi programmi falliscono per una scorretta inizializzazione di certe variabili.
Notiamo infine, nella linea 60, che si stampa N + 1 e non N: questo per mettere in conto l'ultimo tentativo, poiché quando il numero è esatto, non si passa dall'istruzione 35.

Esercizio 4.5 *Come far stampare lo stesso N nella linea 60?*

I cicli *FOR... NEXT*

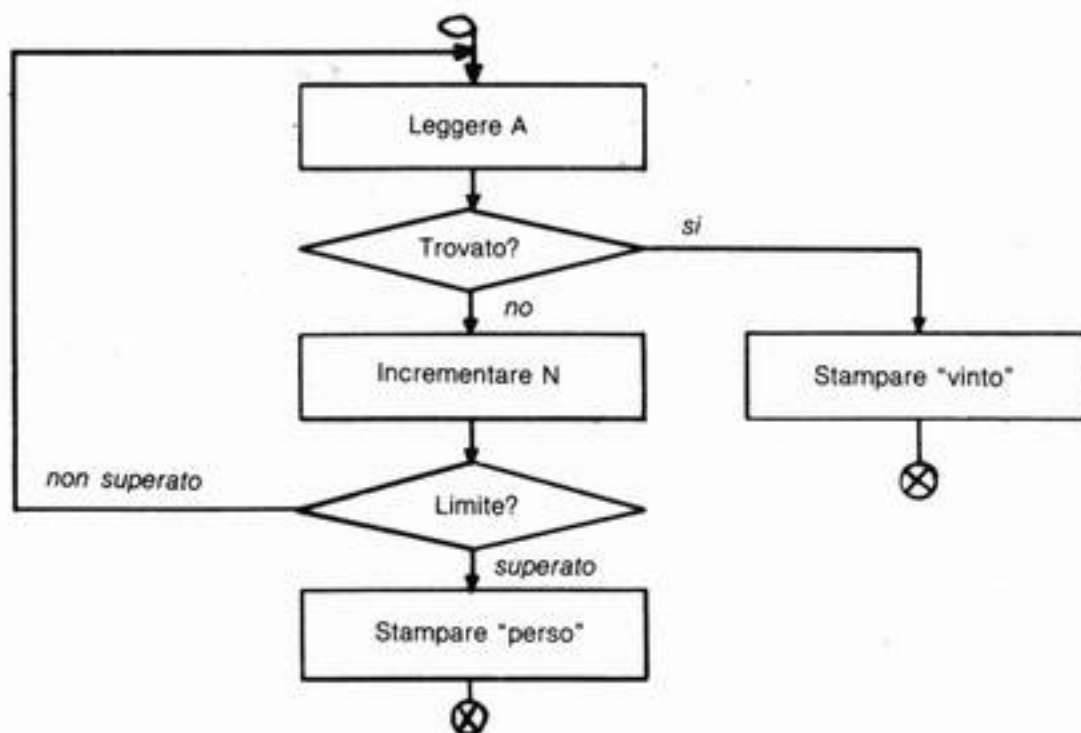
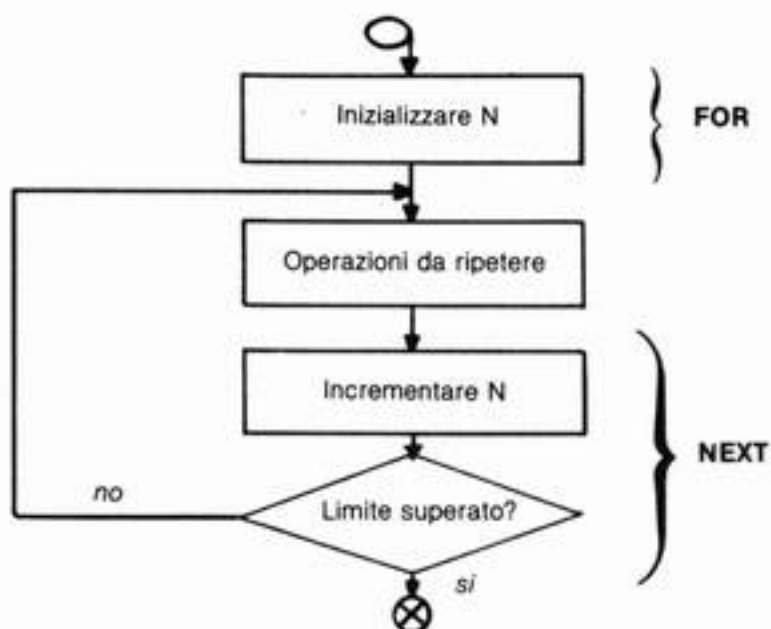
Possiamo ora affrontare un altro difetto del programma nel suo stato attuale: esso permette un numero illimitato di tentativi. È troppo, naturalmente. Dobbiamo autorizzare vari tentativi, ma in numero limitato, ad esempio 5 o 10.
Lo si può fare molto semplicemente. Infatti abbiamo, in ogni momento, una misurazione del numero di tentativi fatti fino a quel punto: è la variabile N che basta testare. Sostituiremo, per farlo, l'istruzione 50 con la sequenza:

```
45 IF N < 10 GOTO 20
50 PRINT "MI DISPIACE. HAI PERSO"
55 END
```

ne segue il programma B.4A:

PROGRAMMA B.4A

```
20 INPUT"INDOVINA UN NUMERO";A
25 E=100*ABS(A-3.25)/3.25
30 IF E<0.5 GOTO 60
40 PRINT"ERRORE";E;"%"
50 PRINT"MI DISPIACE.HAI PERSO"
55 END
60 PRINT"VINTO IN";N+1;"TENTATIVI"
```

Disegnamo lo schema a blocchi corrispondente, senza dettagliarlo troppo:

Se semplifichiamo ancora questo schema a blocchi per farne risaltare l'ossatura, otteniamo:

Questa struttura, molto classica, è del tutto fondamentale dato il suo uso universale: si chiama ciclo. Ma contrariamente ai cicli che abbiamo visto all'inizio, in questo caso, il numero di iterazioni è limitato a

priori: N fa le veci di contatore dei passaggi sulle operazioni da ripetere; gli si dà un valore iniziale, poi si effettua il trattamento da ripetere per i successivi valori di N fintantoché il limite non è superato. Il programma B.4A ci prova che si possono realizzare cicli, in modo del tutto soddisfacente, con le istruzioni che conosciamo già. Eppure, data l'importanza dei cicli, Basic offre un gruppo di istruzioni speciali che permettono di inserirli ancora più facilmente. Si tratta dell'insieme *FOR... NEXT* usato nel programma B.4B.
Si vede quanto sia facile usare tali cicli: basta inquadrare le istruzioni da ripetere (da 20 a 40 nel nostro esempio) con *FOR e NEXT.*

PROGRAMMA B.4B

```
10 FOR N=1 TO 10
20 INPUT"INDOVINA UN NUMERO";A
25 E=100*ABS(A-3.25)/3.25
30 IF E<0.5 GOTO 60
40 PRINT"ERRORE";E;"%"
45 NEXT N
50 PRINT"MI DISPIACE.HAI PERSO"
55 END
60 PRINT"VINTO IN";N+1;"TENTATIVI"
```

— in testa, un'istruzione FOR, della forma:

FOR N= valore di inizio TO valore limite

o, in italiano:

per N= valore di inizio *fino a* valore limite;

— in coda, l'istruzione NEXT N che significa passare al successivo N. Essa incorpora quindi sia l'incremento di N sia il test...
Ovvio, no?

Esercizio 4.6 *Stampare una tabella dei quadrati e delle radici quadrate dei numeri da 1 a 10.*

L'istruzione da ripetere è di stampare, sulla stessa linea, un numero N, il suo quadrato e la sua radice, ossia:

20 PRINT N; N↑2; SQR(N)

Questo bisogna farlo per tutti i valori di N da 1 a 10. Ossia:

10 FOR N = 1 TO 10

e non bisogna dimenticare di terminare con:

30 NEXT N

Esercizio 4.7 *Inserire le istruzioni precedenti, e far eseguire. Cosa manca?*

(Indicazione: non riguarda il ciclo).

Estensioni di *FOR... NEXT*

La forma che abbiamo appena visto non è che un caso particolare della forma più generale:

FOR N = valore di inizio TO valore limite STEP passo

STEP annuncia il passo d'incremento del contatore. Si mette 2, ad esempio, se si vuole che N cresca di 2 in 2;

Esercizio 4.8 *Vogliamo fare la stessa tabella dell'esercizio 4.6, ma soltanto per i valori di N pari.*

Se non mettiamo STEP e un passo, il C64 sottointende un passo uguale ad uno.
I valori degli estremi possono essere qualsiasi e il *passo può essere negativo*. Ad esempio, se volessimo fare la stessa tabella di prima, ma cominciando da 10, poi 9, 8 ecc... scriveremmo:

10 FOR N = 10 TO 1 STEP —1

Non è obbligatorio mettere costanti come estremi: si possono mettere *variabili*, o anche qualsiasi *espressione aritmetica*. Ma le espressioni vengono valutate una volta per tutte quando si entra nel ciclo, anche se l'esecuzione del ciclo provoca un'evoluzione nelle variabili che intervengono. Questo si usa soprattutto nelle scritte della forma:

FOR I = 1 TO N + 1

o

FOR I = 1 TO N STEP 2 * K...

Il *valore limite* che diamo è il valore che non verrà superato per l'esecuzione del ciclo; il test si svolge esattamente come nello schema a blocchi del programma B.4A: il contatore viene modificato e si testa se è ancora compreso tra gli estremi; se lo è, si riesegue il ciclo, se no, abbiamo terminato, e il contatore ha un valore fuori dagli estremi.

Esercizio 4.9 *Per quali valori del contatore vengono eseguiti i cicli seguenti, e qual è il valore finale del contatore?*

10 FOR I = 1 TO 8.5 STEP 2
50 FOR M = 10 TO 3.5 STEP —1

Il trattamento da ripetere per un ciclo può contenere lui stesso un ciclo. Si dice che si hanno *cicli indentati*.

Esercizio 4.10 *Si vuole tracciare una tabella dei numeri da 1 a 12 con i loro quadrati ma con quattro coppie per linea.*
Una soluzione è:

```
10 FOR I=1 TO 9 STEP 4
20 FOR J=0 TO 3
30 PRINT I+J; (I+J)*(I+J),
40 NEXT J
50 NEXT I
```

Il secondo ciclo deve essere completamente contenuto all'interno del primo:

Nota: Se in 30 si scrivesse PRINT I + J; (I + J)↑2, la stampa sarebbe turbata dal fatto che si troverebbe:

$$7^2 = 49.0000001 \text{ e } 9^2 = 81.0000001$$

Perché? Ebbene perché l'interprete calcola x^y nella forma: $e^{y \log x}$ senza distinguere casi particolari per $y = 2$, dal che seguono i suoi errori di arrotondamento.
Si vede in questo caso una delle cause d'inefficienza degli interpreti, mentre un programmatore in linguaggio macchina tiene conto del caso particolare che incontra, e sostituisce la potenza 2 con una moltiplicazione, più veloce.
Si può omettere la ripetizione della variabile che serve come contatore nel NEXT. Così, nell'esercizio 4.6, si sarebbe potuto scrivere 30 NEXT, e nel 4.10, 40 NEXT; 50 NEXT — e il problema dell'ordine non si poneva più.
Nell'esercizio 4.10, avremmo potuto sostituire 40 e 50 con 45 NEXT J, I (attenti all'ordine).
Con la coppia FOR... NEXT, abbiamo appena aggiunto al nostro arsenale uno degli strumenti più efficaci del Basic. Ci rimangono da vedere due altri elementi base per i quali torniamo al nostro programma di gioco.

L'orologio tempo reale

Ciò che manca nel nostro gioco attualmente, è lo "sport". Certo, il giocatore ha diritto ad un numero limitato di tentativi, ma sarebbe molto più spettacolare se il tempo concesso per indovinare il numero giusto fosse limitato.

Il C64 ha il necessario per questo. Infatti, possiede un orologio tempo reale, e del resto è attualmente uno degli unici microcomputer di quella categoria di prezzo che abbiano un tale orologio.
Ma cos'è un orologio tempo reale, e a cosa serve? Naturalmente, un orologio è fatto per dare l'ora, ma come si procede?
Nel caso del C64 l'orologio si comporta come una casella di memoria che si può leggere (è in questo modo che si ottiene l'ora); ma tale casella di memoria ha un comportamento un po' particolare: ogni sesantesimo di secondo, il suo contenuto è aumentato di uno da un processo esterno al microprocessore, che fa intervenire un oscillatore, divisioni di frequenza e interruzioni, e di cui non dobbiamo preoccuparci.
Tutto quello che dobbiamo sapere è che esiste una variabile (riservata) particolare, *TI*, che vale 0 all'accensione e che, in seguito, ha per valore il numero di sessantesimi di secondo trascorsi dall'ultima accensione.
Proviamo il programma:

```
10 ?TI
20 GOTO 10
```

Vedrete un messaggio che non smette di variare, poiché il numero viene aumentato di 1 sessanta volte al secondo.
Le possibilità offerte da questo orologio tempo reale sono molto numerose. Infatti, se scriviamo:

```
10 T1=TI
...
50 T2=TI
```

allora T2-T1 è proporzionale all'intervallo trascorso tra l'esecuzione di 10 e l'esecuzione di 50: abbiamo dunque una misurazione di tale intervallo (per averla in secondi, basta dividere T2-T1 per 60).
Si può, al contrario, generare un intervallo di tempo: supponiamo che tra le istruzioni 10 e 20 si voglia attendere un minuto, si scrive:

```
10...
15 T=TI
16 IF TI-T<3600 GOTO 16
20...
```

In 15, si fissa l'istante di inizio nella variabile T. Poi, in 16, TI rappresenta l'istante attuale. TI-T è l'intervallo trascorso tra 15 e l'esecuzione attuale di 16: esso cambia ogni volta dato che il tempo passa. Fintantoché l'intervallo di tempo è minore di 3600 scatti = 60 secondi = 1 minuto, si riesegue 16. Ma l'intervallo cresce senza pausa: finirà con l'essere di un minuto, e allora si passerà a 20.
Va bene, ma come avere l'ora nella forma usuale? Si fa appello ad un'altra variabile riservata del C64, TI$ che, come vedremo più tardi

è una variabile stringa di caratteri (il suo nome termina con $). Essa contiene l'ora sotto forma di una stringa di 6 caratteri, della forma hhmmss:
?TI$ farà stampare, ad esempio, 102308 se sono le ore 10, 23 minuti, 08 secondi.
Ma, per questo, bisogna mettere il C64 all'ora giusta, altrimenti considera che all'accensione sono le ore 0, 0 min, 0 s.
La regolazione dell'ora si fa con un assegnamento del tipo:

TI$ = "101000" per dire che sono le ore 10 e 10 min.

Una volta che abbiamo l'ora in questo modo, possiamo chiedere al C64 di accendere una lampada ad una certa ora e di spegnerla un po' più tardi. Gli possiamo chiedere di togliere la corrente in tutti gli uffici dopo le 18, ma dopo le 12 al sabato, e pure di tenere conto dei giorni festivi.
Sarà quindi questo orologio tempo reale a servirci per limitare il tempo concesso al giocatore.
Come? È semplicissimo. Annoteremo, all'inizio del gioco, l'ora nella variabile T:

5 T = TI

poi, ad ogni tentativo del giocatore, testeremo se il tempo non è superato:

15 IF (TI-T)/60 > 120 GOTO 50

Esercizio 4.11 *Qual è il tempo concesso al giocatore dall'istruzione 15?*

Si giunge quindi al programma B.5.

PROGRAMMA B.5

```
5 T=TI
10 FOR N=1 TO 10
15 IF (TI-T)/60>120 GOTO 50
20 INPUT"INDOVINA UN NUMERO";A
25 E=100*ABS(A-3.25)/3.25
30 IF E<0.5 GOTO 60
40 PRINT"ERRORE";E;"%"
45 NEXT N
50 PRINT"MI DISPIACE.HAI PERSO"
55 END
60 PRINT"VINTO IN";N;"TENTATIVI E";INT((TI-T)/60);"SECONDI
```

Abbiamo pure incorporato nella linea 60 la stampa del tempo impiegato dal giocatore per trovare la soluzione.

Esercizio 4.12 *Ricostruire lo schema a blocchi del programma B.5.*

Esercizio 4.13 *Modificare l'istruzione 60 per stampare il tempo al 1/100mo di secondo (come per le gare di sci).*

Rimane da apportare un piccolo perfezionamento: quando stampiamo "perso", sarebbe un bene distinguere se si tratta di superamento del tempo o di numero troppo grande di tentativi. È lo scopo dell'esercizio seguente.

Esercizio 4.14 *Quando il giocatore ha perso, stampare la causa del fallimento, tempo o numero di tentativi.*

Istruzione STOP, tasto 'STOP', comando CONT

Il nostro programma di gioco è ormai giunto ad un buon livello di complessità. Di conseguenza, possono presentarsi difficoltà per effettuare la messa a punto. Come ci aiuta a risolverle il C64?
Il primo strumento è costituito dai messaggi d'errore stampati dal C64 in caso di situazione anormale. Ad esempio, se mai richiamaste la funzione SQR con un argomento negativo, avreste il messaggio:

?ILLEGAL QUANTITY ERROR IN numero d'istruzione

poi, READY viene visualizzato, il che indica che il C64 è pronto ad accettare un comando in modo diretto. Notiamo, anche qui, che il comando occupa due linee.
Il comando diretto più giudizioso da inserire in un caso simile è PRINT certe variabili. Infatti nel caso di un'interruzione di questo tipo, tutte le variabili del programma sono conservate, potete quindi chiedere la loro stampa in modo diretto. Ad esempio, se la nostra radice quadrata ad argomento negativo dipende da una variabile X, batteremo ?X. Allora vediamo che X non ha il valore che prevedevamo. Possiamo chiedere allora il LIST dell'istruzione che calcola X. Vediamo quindi che manca un'operazione, e siamo pronti a correggere l'istruzione.
Attenzione, dal momento in cui operiamo la minima modifica sul programma, le variabili non sono più conservate. Ci conviene quindi esaminare più variabili possibili prima di iniziare le correzioni.
Tale è lo svolgersi abituale della correzione degli errori. I messaggi di errore sono listati in appendice, con un tentativo di analisi delle loro cause più frequenti.
Il procedimento è diverso se non si presenta alcun errore che susciti un messaggio mentre i risultati sono sbagliati. Come fare in tal caso? Allora suddivideremo il programma in piccole tappe, tra le quali inseriremo istruzioni STOP. Ad esempio, se un programma ha due tappe, la prima da 10 a 50 e la seconda da 60 a 150, intercaleremo un 55 STOP.

Quando si arriverà in 55, il C64 stamperà:

BREAK IN 55
READY.

e si fermerà.
Notiamo già che il fatto di ottenere una stampa tale significa che la prima tappa si è svolta fino in fondo. Correttamente? Per saperlo, poiché il C64 è fermo, vi basta chiedere la stampa diretta delle variabili strategiche.
Supponiamo che tutto sia corretto. Vorremmo ora eseguire la seconda tappa. Ebbene, per fare questo, disponiamo del comando CONT, che vuol dire "continuate, ora ho fatto quello che volevo al momento dell'arresto". Attenti, non potete usare CONT se, durante l'arresto, avete modificato il programma.
C'è ancora una cosa possibile. Supponiamo che, nell'esempio qui sopra, non otteniamo mai il messaggio BREAK IN 55. Ciò significa che, durante la prima tappa, il programma entra in un ciclo senza fine. Come sapere a che punto siamo? Basta premere il tasto 'Run/Stop' (senza 'SHIFT'). Questo tasto simula un'istruzione STOP nella linea in corso di esecuzione al momento in cui si preme: otteniamo il messaggio BREAK IN 25 (ad esempio). Rilanciamo l'esecuzione mediante CONT. L'esame di qualche variabile e un listing della zona del programma indicato dal BREAK IN permettono, normalmente, di scoprire l'errore.
Infine, un ultimo strumento di soccorso è semplicemente di aggiungere, ad intervalli regolari, la stampa delle principali variabili; basterà poi, quando il programma sarà messo a punto, sopprimere le istruzioni di stampa superflue.

Domanda: *qual è la differenza tra STOP e END? L'unica differenza è che STOP fa stampare il messaggio BREAK IN... mentre END non lo fa. Si può pure ricominciare con CONT dopo un'istruzione END.*

RICAPITOLAZIONE

Questo capitolo ci ha permesso di vedere gli strumenti base del programmatore:

- *le istruzioni fondamentali **IF** e **FOR**;*
- *la manipolazione dell'**orologio tempo reale** del C64;*
- *qualche **aiuto alla messa a punto** dei programmi.*

Siamo ora pronti per affrontare le tecniche elaborate di programmazione.

Capitolo 5

Programmazione evoluta

Istruzioni *DATA*, *READ* e *RESTORE*

Il nostro programma di gioco ci condurrà ora a tecniche più sofisticate. Supponiamo che si voglia giocare in tanti. Quindi ora abbiamo bisogno di una stringa di numeri da indovinare. Infatti, se il giocatore n. 3 ha visto che i due precedenti dovevano indovinare 3.25, non dovrà spremersi troppo il cervello! Il programma B.6A dà una soluzione. Per semplificare abbiamo soppresso il limite sul numero di tentativi autorizzati ma, naturalmente, abbiamo lasciato la limitazione nel tempo.

Attenti: se avete lasciato in memoria il precedente programma B.5, vi conviene cancellare la memoria con il comando NEW prima di battere il programma qui sotto. Infatti abbiamo rinumerato le linee e l'inserimento sovrapposto di B.6A su B.5 dà un "caos" inintelligibile. Questa rinumerazione ha lo scopo di ridare un po' di aria al programma per permettere future aggiunte e modifiche.

PROGRAMMA B.6A

```
10 READ C
20 T=TI
30 IF TI-T>7200 GOTO 90
40 INPUT"CHE NUMERO PROPONI";A
50 E=100*ABS(A-C)/C
60 IF E<0.5 GOTO 110
70 PRINT"ERRORE";INT(E*100)/100;"%"
80 GOTO 30
```

```
90 PRINT"TROPPO TARDI! CEDI IL TUO POSTO AL GIOCATORE
                                          SUCCESSIVO"
100 GOTO 10
110 PRINT"VINTO IN";INT((TI-T)/.6)/100;"SECONDI"
120 GOTO 10
200 DATA3.25,7.65,2,121,449,0.075,18
210 DATA5.8,210,78.31,901.5,31,4.18,2.7
```

Due nuove istruzioni appaiono in questa versione: *READ* e *DATA*.
DATA serve semplicemente per specificare una lista di costanti separate da virgole; l'istruzione DATA viene considerata soltanto in relazione con una istruzione READ; altrimenti è "trasparente" per il programma.
Si può mettere un'istruzione DATA in qualsiasi posto del programma; quando Basic ci arriva sopra, non fa niente e passa all'istruzione successiva. Così, 200 avrebbe potuto essere numerato 55, e 210 avrebbe potuto essere numerato 75, ad esempio. L'esecuzione sarebbe passata lo stesso direttamente da 50 a 60 e da 70 a 80. I dati presi non sarebbero stati alterati. Ciò che ha importanza è l'ordine delle diverse istruzioni DATA e l'ordine dei dati all'interno di una stessa istruzione DATA.
Al momento della partenza del programma (subito dopo il RUN), il primo dato del primo DATA verrà preso al momento del primo READ eseguito. Poi, con il susseguirsi delle esecuzioni dei READ successivi, il secondo dato dal primo DATA, poi il terzo ecc. poi il primo dato del secondo DATA e così via.
Nel nostro esempio, siccome c'è un READ per ogni nuovo giocatore, i numeri successivi da cercare sarebbero 3.25, poi 7.65, poi 2 ecc... ossia 14 possibilità diverse concesse.
Cosa avviene se ci sono più di 14 giocatori? Ebbene, c'è un errore: se si prova un READ mentre l'ultimo dato dell'ultimo DATA è stato "letto", il messaggio ?OUT OF DATA ERROR IN... viene visualizzato dal C64.
L'istruzione *RESTORE* ci permette di aggirare l'ostacolo: essa dà, infatti, la possibilità di ritornare all'inizio dei DATA. Cioè, dopo un RESTORE, un READ ottiene di nuovo il primo dato del primo DATA poi, ecc. In questo caso, ripercorriamo quindi la stessa serie di numeri da indovinare ogni 14 giocatori.
Per questo, ci serve una variabile J, che conterrà il numero di giocatori. Quando tale numero diverrà 14 o divisibile per 14, bisognerà fare un RESTORE.
Ma, come si vede che X è divisibile per Y? Semplicissimo: se X è divisibile per Y, il quoziente X/Y è uguale a INT(X/Y) dato che è intero.
Ed ecco il programma B.6B:

PROGRAMMA B.6B

```
10 J=J+1 :PRINT"GIOCATORE NO.";J
20 READ C:T=TI
30 IF TI-T>7200 GOTO 70
40 INPUT"CHE NUMERO PROPONI";A
50 E=100*ABS(A-C)/C :IF E<0.5 GOTO 90
60 PRINT"ERRORE";INT(E*100)/100;"%":GOTO 30
70 PRINT"TROPPO TARDI! CEDI IL TUO POSTO AL GIOCATORE
                                               SUCCESSIVO"
75 IF INT(J/14)=J/14 THEN RESTORE
80 GOTO 10
90 PRINT"VINTO IN";INT((TI-T)/.6)/100;"SECONDI"
100 GOTO 75
200 DATA3.25,7.65,2,121,449,0.075,18
210 DATA5.8,210,78.31,901.5,31,4.18,2.7
```

Una variante supplementare, rispetto al programma B.6A, giunge dal fatto che vi sono più istruzioni su certe linee. Si possono, in effetti, mettere più istruzioni per linea a condizione di separarle con il carattere due-punti (:). Ciò rende i programmi più compatti, ma anche meno leggibili.
Naturalmente, se la linea "1" racchiude più istruzioni, un GOTO 1 porterà alla prima, non in mezzo alla linea! Quindi se un'istruzione deve essere meta di un GOTO o di un IF, deve essere sola sulla propria linea, o in testa alla linea.
Altra variante, il numero 5.8 si è spostato dall'inizio del secondo DATA alla fine del primo. Ciò non cambia assolutamente nulla all'ordine dei dati di cui verrà tenuto conto.

Esercizio 5.1 *Un altro metodo per evitare l'esaurimento dei dati sarebbe di aggiungere, alla fine della serie, un dato finto diverso dai numeri che vogliamo trattare, ad esempio 99999, e che testiamo: se lo troviamo facciamo RESTORE. Componete una versione del programma che utilizzi questo metodo.*

DIM e *array*

Affrontiamo ora una nozione importante che aumenterà fortemente la potenza di trattamento messa a nostra disposizione.
Giochiamo sempre con più giocatori, che limitiamo a 14, ad esempio.
Quello che vogliamo in aggiunta è che, una volta effettuata la partita da tutti i giocatori, un quadro riassuntivo dei punteggi ottenuti, in numero di secondi, venga visualizzato.

Perché ciò avvenga, dobbiamo introdurre una nuova variabile: SC. Otteniamo allora il programma seguente:

```
10 FOR J=1 TO 14 : PRINT"GIOCATORE N.";J
20 READ C : T=TI
30 SC=(TI-T)/60 : IF SC>120 GOTO 70
40 INPUT"NUMERO PROPOSTO";A
50 E=100*ABS(A-C)/C : IF E<0.5 GOTO 90
60 PRINT"ERRORE" ; INT(E*100)/100 ; "%" : GOTO 30
70 PRINT" TROPPO TARDI !  LASCIATE IL POSTO AL  GIOCATORE
                                                SUCCESSIVO"
80 GOTO 100
90 PRINT"VINCE IN" ; INT(SC*100)/100;"SECONDI"
100 NEXTJ
110 PRINT"PUNTEGGIO=" ; SC : END
200 DATA 3.25,7.65,2.121,499,0.075,18,5.8
210 DATA 210,78.31,901.5,31,4.18,2.7
```

Ma questa soluzione non è soddisfacente. Infatti, questo programma non stamperà mai altro che il punteggio dell'ultimo giocatore. Ciò che ci serve è un punteggio per ogni giocatore, cioè 14 variabili simili, collegate ciascuna ad un giocatore J.
Basic ha uno strumento per questo scopo. Permette, infatti, che SC sia una variabile multipla di cui SC(1) sarà il primo valore, SC(2) il secondo, ecc.
Il numero dell'elemento desiderato viene messo tra parentesi: si chiama l'indice. L'indice può essere una variabile: SC(I) è l'Iesimo elemento, o anche una espressione: SC(3 * I-4).
Una simile variabile multipla si chiama un *array*. Dobbiamo avvertire Basic del fatto che una variabile è un array e del numero di elementi (questo per conservare posto in memoria). Lo si fa con un'istruzione DIM. Per il nostro esempio, abbiamo: DIM SC(14). In realtà qui riserviamo posto per 15 elementi, poiché è utilizzabile l'indice 0.
Naturalmente, l'istruzione DIM deve essere eseguita prima di qualsiasi operazione sulla variabile in questione; in compenso, deve essere eseguita una volta sola.
L'istruzione DIM non è necessaria fintantoché il valore massimo dell'indice non supera 10: infatti, il C64 riserva automaticamente posto per 10. Se la dimensione dell'array deve essere minore di 10, l'istruzione DIM è utile lo stesso perché libera del posto.
Più array possono essere dimensionati mediante una stessa istruzione: DIM A(25), B(50).
Il valore massimo assegnato all'indice può essere una variabile (che è stata appena inizializzata): DIM A(N).
Usando tali proprietà, arriviamo al programma B.7.

PROGRAMMA B.7

```
5 DIM SC(14)
10 FOR J=1 TO 14 :PRINT"GIOCATORE NO.";J
20 READ C:T=TI
30 SC(J)=(TI-T)/60 :IF SC(J)>120 GOTO 90
40 INPUT"NUMERO PROPOSTO";A
50 E=100*ABS(A-C)/C :IF E<0.5 GOTO 90
60 PRINT"ERRORE";INT(E*100)/100;"%":GOTO 30
70 PRINT"TROPPO TARDI! CEDI IL TUO POSTO AL GIOCATORE
                                          SUCCESSIVO"
80 GOTO 100
90 PRINT"VINTO IN";INT(SC(J)*100)/100;"SECONDI"
100 NEXT J
110 PRINT "GIOCATORE","PUNTEGGIO" :PRINT
120 FOR I=1TO14
130 PRINT I,INT(SC(I)*100)/100 :NEXT I
200 DATA3.25,7.65,2,121,449,0.075,18
210 DATA5.8,210,78.31,901.5,31,4.18,2.7
```

Lasciamo ora da parte il nostro gioco per qualche istante per vedere diversi complementi sugli array.
Vi conviene salvare questo programma su cassetta mediante l'ordine SAVE. Poi potrete cancellare la memoria con NEW, il vostro C64 sarà disponibile per qualche esercizio.

Riempimento di un array

L'istruzione di inserimento INPUT può essere messa in un ciclo del tipo: FOR I = 1 TO 10: INPUNT A(I): NEXT I. Ci piacerebbe sapere ad ogni istante quale elemento deve essere introdotto e avere messaggi stampati come:

A(1)?
A(2)? ecc.

Siccome il messaggio comprende un elemento variabile (il valore dell'indice), la forma INPUT "TESTO";... non è adeguata. Bisogna usare un PRINT che termini con punto e virgola (;) per battere il valore sulla stessa linea:

```
10 FOR I=1 TO 10
20 PRINT "A("; I")";
30 INPUT A(I)
40 NEXT I
```

Somma e media degli elementi di un array

Gli elementi di un array possono rappresentare le varie osservazioni statistiche di una grandezza, ad esempio, l'altezza dei vari alunni di una classe. La prima operazione statistica da effettuare su di una di-

stribuzione è calcolare la media; per questo, bisogna calcolarne prima la somma.
Per calcolare tale somma, useremo una variabile S inizializzata a 0 e alla quale, in un ciclo, verrà sommato in successione ciascuno degli elementi:

```
10 DIM A(N)
20 REM NORMALMENTE QUI AVVIENE LA LETTURA DEGLI
   ELEMENTI
30 S=0
40 FOR I=1 T0 N
50 S=S+A(I)
60 NEXT I
70 M=S/N
80  PRINT "SOMMA="; S, "MEDIA="; M
```

L'inizializzazione di S a 0 viene effettuata in trenta; tale inizializzazione è superflua all'inizio di un programma; può essere necessaria se si arriva in 10 dopo aver fatto altre operazioni.
In 20 appare una nuova istruzione: *REM* (abbreviazione di "remarque": nota). Essa non influisce in alcun modo sul funzionamento del programma, ma permette di incorporarvi commenti esplicativi, cosa spesso utile (naturalmente, tali commenti usano posto memoria).

Esercizio 5.2 *Calcolare la varianza dell'insieme degli elementi A qui sopra:*

$$V = \frac{\Sigma_i (A_i - M)^2}{N-1}$$

Gli arrays sono particolarmente indicati per rappresentare vettori di uno spazio vettoriale su K. Ad esempio A(1), A(2), A(3) saranno le tre componenti del vettore $\overline{A}$ nello spazio a tre dimensioni. Vedremo più tardi che Basic permette pure la manipolazione di matrici.

Esercizio 5.3 *Dati i due vettori U e V di uno spazio ad N dimensioni, calcolare il loro prodotto scalare* ($\Sigma_i u_i v_i$).

Numeri casuali - Funzione *RND*

È ora giunto il momento di risolvere uno dei problemi che maggiormente ci aveva preoccupato all'inizio del nostro gioco: come fare perché il giocatore non possa, anche se bara, trovare in anticipo il numero da indovinare?
Il meglio è che il computer stesso non conosca tale numero in anticipo, cioè che lo sorteggi al momento di usarlo. Va bene, ma come può un computer, che deve avere il comportamento più deterministico e più prevedibile possibile, dare numeri casuali?

A priori, ciò sembra nocivo. Ebbene, ed è paradossale, esistono algoritmi che richiamano calcoli ben determinati, che danno quello che si chiama serie pseudocasuale, cioè successioni di numeri ben determinati, ma aventi proprietà statistiche tali che si possa considerare che tutto si svolge come se i numeri fossero stati sorteggiati.
In questo contesto, il termine "numero casuale" non può riferirsi ad un numero isolato, è solo a livello di una successione (numerosa) di numeri che esso ha un senso.
A cosa serviranno mai tali successioni? Servono per calcoli di simulazione nei quali bisogna tener conto di fenomeni aleatori. Ad esempio, supponiamo di voler simulare dieci anni di sfruttamento di una proprietà agricola.
Uno degli elementi che intervengono può essere la produzione di grano di un certo campo, sappiamo che, qualsiasi cosa avvenga, questa produzione è compresa tra 10 t (annata dalle condizioni atmosferiche sfavorevoli, cattive semenze, ecc.) e 15 t (annata che unisce eccezionalmente tutti i fattori favorevoli).
Per simulare i rischi dovuti a cause poco note, la cosa migliore è, per ogni anno, di sorteggiare un numero tra 10 e 15: sarà il modo migliore per ottenere, nella nostra simulazione su dieci anni, un certo numero di buone annate e di cattive annate, che possa rappresentare la realtà.
In realtà, in questo caso, non sorteggeremo un numero distribuito uniformemente tra 10 e 15: cercheremo di produrre una legge di probabilità ottenuta da osservazioni o rispondente ad un modello.
Si vede che i numeri casuali possono essere molto utili. Il C64 può fornircene. Basta fare Y = RND(X) per ottenere un numero casuale tra 0 e 1.
Se X > 0, si generano diverse sequenze pseudocasuali: successivi richiami con lo stesso valore di X > 0 danno gli elementi successivi di una stessa sequenza (il numero ottenuto cambia ad ogni richiamo, esso fa parte della stessa sequenza). Cambiando il valore di X, si cambia successione.
Essendo il numero ottenuto compreso tra 0 e 1, cioè della forma 0.232745, si dovrà fargli subire una trasformazione per rispondere al nostro problema.
Generalmente, bisogna ottenere una serie di numeri Y compresi tra due valori A e B. La successione di operazioni seguente permette di ottenere tali numeri:

```
Y = A + (B—A) * RND(1)
```

il più delle volte, si userà RND(1).
Applichiamo ciò nel programma B.8 per ottenere un numero compreso tra 1 e 100:

PROGRAMMA B.8

```
5 INPUT"NUMERO DI GIOCATORI";N:DIM SC(N)
10 FOR J=1 TO N:PRINT"GIOCATORE NO.";J
20 C=1+99*RND(1):T=TI
30 SC(J)=(TI-T)/60 :IF SC(J)>120 GOTO 90
40 INPUT"NUMERO PROPOSTO";A
50 E=100*ABS(A-C)/C :IF E<1 GOTO 90
60 PRINT"ERRORE";INT(E*100)/100;"%":GOTO 30
70 PRINT"TROPPO TARDI! CEDI IL TUO POSTO AL GIOCATORE
                                          SUCCESSIVO"
80 SC(J)=0 :GOTO 100
90 PRINT"VINTO IN";INT(SC(J)*100)/100;"SECONDI"
95 SC(J)=1+INT((120-SC(J))/24)
100 NEXT J
110 PRINT "GIOCATORE","PUNTEGGIO" :PRINT
120 FOR I=1TO N
130 PRINT I,INT(SC(I)*100)/100 :NEXT I
```

Il cambiamento più notevole sta quindi nell'istruzione 20 dove il numero da cercare viene ora sorteggiato. Lo si potrebbe rendere ancora più casuale (infatti, alla partenza del gioco, si avrà sempre la stessa serie di numeri) indicizzandolo sul tempo, ad esempio scrivendo:

```
7 T = TI; X = T—100*INT(T/100)
20 C = 1 + 99*RND(1 + X):T = TI
```

La linea 7 ha l'effetto di mettere in X le due cifre di destra di TI. Se TI = 2873, INT(TI/100) è uguale a 28 e X sarà uguale a 2873—2800 ossia 73.
Così stando le cose, a seconda del tempo trascorso dall'accensione, verrà generata una serie diversa. Sono stati introdotti altri miglioramenti che hanno facilitato leggermente il gioco. Ad esempio, l'approssimazione è stata messa ad 1%, ma siete perfettamente liberi di modificare questa approssimazione e il limite di tempo.
Il numero N dei giocatori è ormai variabile.
Il punteggio viene dato in punti da 0 a 5 (0 se il giocatore non ha trovato in tempo, 5 se ha trovato prestissimo).

Esercizio 5.4 *Misurare la bontà statistica del generatore di numeri casuali del C64.*

Per questo, tireremo 1000 numeri a sorte, compresi tra —1 e +1. Calcoliamo poi media e varianza (valori ideali 0 e 1/3) così come gli effettivi delle 10 classi:

[—1,—4/5];[—4/5,—3/5];...;[0,1/5]...[4/5,1] (valore ideale 100).

Il programma seguente è una delle possibili soluzioni:

```
10 FOR I=1 TO 1000
20 N=(-1)+2*RND(1)
30 S=S+N:S2=S2+N↑2
40 J=1+INT((N+1)*5)
50 C(J)=C(J)+1
60 NEXT I
70 M=S/1000:V=(S2-1000*M↑2)/999
80 PRINT"MEDIA =";M,"VARIANZA =";V
90 PRINT"CLASSI"
```

Attenzione, non succede niente per 1 minuto e mezzo. Siate pazienti!...

Array multidimensionali

Perfezioniamo il nostro gioco, di modo che permetta ad ogni giocatore di disputare diverse partite e che visualizzi i punteggi di ciascuna di queste partite.
Il C64 permette di formare degli array a doppia entrata (detti anche array a due indici, o a due dimensioni).
Per fare ciò usiamo un'istruzione DIM della forma:

DIM SC(NJ,NP)

se ci sono NJ giocatori e NP partite. Il punteggio del giocatore J nella partita P verrà designato con SC(J,P); ne segue il programma B.9.

PROGRAMMA B.9

```
10 INPUT"NUMERO DI GIOCATORI";NJ
20 INPUT"QUANTE PARTITE";NP
30 DIM SC(NJ,NP)
40 FOR P=1 TO NP:FOR J=1 TO NJ
50 PRINT"GIOCATORE NO.";J;"PARTITA";P
60 C=1+99*RND(1):T=TI
70 SC(J,P)=(TI-T)/60:IF SC(J,P)>120 GOTO 110
80 INPUT"NUMERO PROPOSTO";A
90 E=100*ABS(A-C)/C :IF E<1 GOTO 130
100 PRINT"ERRORE";INT(E*100)/100;"%":GOTO 70
110 PRINT"TROPPO TARDI!CEDI IL POSTO AL GIOCATORE
                                       SUCCESSIVO"
120 SC(J,P)=0 :GOTO 150
130 PRINT"VINTO IN";INT(SC(J,P)*100)/100;"SECONDI"
140 SC(J,P)=1+INT((120-SC(J,P))/24)
```

```
150 NEXT J,P :PRINT :PRINT
160 PRINT "GIOCATORE", :FOR P=1 TO NP
170 PRINT "PARTITA";P; :NEXT P :PRINT :PRINT
180 FOR J=1 TO NJ :PRINT J,
190 FOR P=1 TO NP :PRINT SC(J,P),
200 NEXT P:PRINT :NEXT J
```

In matematica, questi arrays rettangolari si chiamano matrici i cui elementi sono distribuiti in linee e colonne. Le matrici hanno numerose applicazioni che sono quindi realizzabili sul C64.

Esercizio 5.5 *Scrivere un programma che calcoli il prodotto C di due matrici A e B (richiamo della formula di definizione:* $C_{ij} = \Sigma_k a_{ik} \cdot b_{kj}$*).*

Ciò che abbiamo appena esposto riguardo agli arrays può essere generalizzato: il numero delle dimensioni può essere qualsiasi; in tre dimensioni, gli elementi si distribuiscono in piani, linee e colonne. L'unica restrizione formale al numero delle dimensioni è che l'istruzione DIM deve limitarsi a 80 caratteri. Ma altre limitazioni intervengono prima: non deve essere superata la grandezza della memoria rimasta a disposizione.

Manipolazione di stringhe di caratteri

Ci rimane ancora un miglioramento da apportare al nostro programma. Infatti, sarebbe desiderabile che, al momento della stampa dei risultati, venisse scritto il nome di ogni giocatore, e non il suo numero. Vedremo che questo è possibile. È infatti indispensabile che il computer permetta di manipolare testi o nomi. In gestione, bisogna pur manipolare il nome dei clienti!
Il C64 ammette due tipi di variabili: le variabili numeriche, che già conosciamo, e le variabili alfanumeriche, o stringhe di caratteri. Il nome di una variabile stringa viene formato come quello di una variabile numerica, aggiungendo un $ alla fine:

A: variabile numerica
A$: variabile alfanumerica

Congiuntamente, nello stesso programma, possono essere utilizzate le variabili distinte A ed A$.
Soltanto i due primi caratteri del nome di una variabile sono considerati (a parte il $): AL$ e ALFA$ sono la stessa variabile.
È possibile formare arrays di stringhe di caratteri: DIM NOM$(N) riserva un array di N stringhe di caratteri, e ciascun elemento può contenere un numero variabile di caratteri.

Per assegnare un valore ad una stringa di caratteri, l'istruzione più semplice è l'assegnamento classico: A$ = "BUONGIORNO".
Si noti che il valore assegnato è racchiuso tra apici.
Si può anche leggere la variabile dalla tastiera con INPUT A$. In tal caso non è necessario mettere BUONGIORNO tra apici, poiché il C64 si aspetta una stringa di caratteri.
Si possono usare, infine, READ e DATA. Nel DATA le stringhe sono normalmente senza apici, tranne se contengono un carattere speciale come spazio, virgola, due punti o i caratteri grafici:

```
10 READ A$, B$
20 DATA BUONGIORNO, "A PIÙ TARDI"
```

Siamo ora pronti per capire il programma B.10.

PROGRAMMA B.10

```
10 INPUT"NM.DI GIOCATORI,NM.DI PARTITE";NJ,NP
20 DIM NOM$(NJ),SC(NJ,NP)
30 FOR J=1 TO NJ:PRINT"NOME DEL GIOCATORE NO.";J;
35 INPUT NOM$(J) :NEXT J
40 FOR P=1 TO NP:FOR J=1 TO NJ
50 PRINT"GIOCATORE NO.";J;"(";NOM$(J);")","PARTITA";P
60 C=1+99*RND(1):T=TI
70 SC(J,P)=(TI-T)/60:IF SC(J,P)>120 GOTO 110
80 INPUT"NUMERO PROPOSTO";A
90 E=100*ABS(A-C)/C :IF E<1 GOTO 130
100 PRINT"ERRORE";INT(E*100)/100;"%":GOTO 70
110 PRINT"TROPPO TARDI!CEDI IL POSTO AL GIOCATORE
                                          SUCCESSIVO"
120 SC(J,P)=0 :GOTO 150
130 PRINT"VINTO IN";INT(SC(J,P)*100)/100;"SECONDI"
140 SC(J,P)=1+INT((120-SC(J,P))/24)
150 NEXT J,P :PRINT :PRINT
160 PRINT "GIOCATORE",:FOR P=1 TO NP
170 PRINT "PARTITA";P;:NEXT P :PRINT :PRINT
180 FOR J=1 TO NJ :PRINT NOM$(J),
190 FOR P=1 TO NP :PRINT SC(J,P),
200 NEXT P:PRINT :NEXT J
```

Nel precedente programma non facciamo altro che leggere un nome, riporlo in NOM$(J) per memorizzarlo e stamparlo un po' più tardi. È spesso l'unico trattamento da effettuare sulle stringhe di caratteri. Ma, in alcuni casi, si devono effettuare trattamenti sulle stringhe, come confronti, estrazioni, conversioni, ecc. Vediamo ora le operazioni di questo tipo disponibili sul C64.

Confronti: un'istruzione come IF A$ = B$ GOTO... permette di confrontare le due stringhe A$ e B$. Le applicazioni sono numerose: si

può, ad esempio, verificare se una parola appartiene ad un dizionario o se un nome figura in una lista di persone autorizzate... Un altro confronto come IF A$ < B$... è pure utile: infatti, la parola A$ è considerata minore di B$ se la precede nell'ordine alfabetico, dal che segue un modo per classificare liste di nomi in ordine alfabetico.

Concatenamento: l'operatore + applicato a due stringhe di caratteri ne produce il concatenamento:

"BUON" + "GIORNO" dà BUONGIORNO. Battete
A$ = "GIO"
B$ = "NO"
C$ = "BUON" + A$ + "R" + B$
?C$ di nuovo, ottenete BUONGIORNO

Stringa vuota: è la stringa di 0 caratteri. A$ = "" dà ad A$ il valore 'stringa vuota'; per ogni X$, X$ + A$ sarà identica a X$.

Estrazione di sottostringhe: il C64 possiede un certo numero di funzioni che permettono la "manipolazione" delle stringhe di caratteri. Se il nome della funzione termina con $, il suo risultato è una stringa di caratteri; altrimenti il risultato è un numero.

— *LEN(X$):* fornisce la lunghezza (numero di caratteri) della stringa X$.

— *LEFT$(X$,N):* fornisce gli N caratteri più a sinistra, estratti dalla stringa X$.

— *RIGHT$(X$,N):* fornisce gli N caratteri più a destra, estratti dalla stringa X$. Se N > LEN(X$), otteniamo tutta la stringa (questo vale anche per LEFT$);

— *MID$:* estrae caratteri in mezzo ad una stringa; può avere due o tre argomenti;

— *MID$(X$,K):* fornisce i caratteri estratti dalla stringa X$ dalla posizione K in poi. Se K > LEN(X$), otteniamo la stringa vuota;

— *MID$(X$,K,N):* fornisce la sottostringa di N caratteri estratti da X$ dal Kesimo in poi. Se K > LEN(X$), otteniamo la stringa vuota; se N richiede più caratteri di quanti ne rimangano in X$, otteniamo tutti i caratteri di X$ dal Kesimo in poi.

Esercizio 5.6 *Sostituire il quinto carattere della stringa X$ con la lettera A.*

Esercizio 5.7 *Verificare che la stringa A$ contenga la sottostringa B$. Mettere nella variabile K, 0 se non la contiene, e se la contiene, la prima posizione in A$ in cui si trova B$.*

Ad esempio, se B$ = "BRA", in ABRACADABRA, troviamo B$ in 2 e in 9, e dobbiamo ottenere K = 2. Con BUONGIORNO, dobbiamo ottenere K = 0.

Funzioni di conversione: le altre funzioni stringhe di caratteri effettuano conversioni tra i numeri e la loro rappresentazione sotto forma di stringhe di caratteri o di codice ASCII (il codice ASCII è il modo di rappresentazione interna scelto dal C64 e la maggior parte dei microcomputers: ogni carattere è rappresentato da un valore binario scelto, che occupa un byte).
Vi sono quattro funzioni di questo tipo:

— *ASC(X$):* fornisce il valore in decimali del byte che rappresenta in ASCII il primo carattere di X$. Esempio: ?ASC("A") dà 65.

— *CHR$(K):* fornisce una stringa di un carattere: il carattere di cui K è il codice ASCII. Esempio: ?CHR$(65) fa stampare una A.

— *STR$(A):* fornisce la stringa di caratteri che è la rappresentazione decimale del numero A. Se A = 3.5 STR$(A) è la stringa: spazio (per il segno), 3, punto, 5. ?A e ?STR$(A) producono la stessa stampa con la sola differenza che ?A è seguito da un movimento del cursore verso destra e non ?STR$(A). Osservate la differenza tra:

?"AA";A;"AAA" e ?"AA";STR$(A);"AAA"

C'è tuttavia un'importante differenza: STR$(A) è una stringa di caratteri che può essere soggetta ad operazioni LEFT$...

— *VAL(X$):* fornisce il valore del numero del quale X$ è la rappresentazione decimale. Gli unici caratteri permessi in X$ sono le cifre, il punto, lo spazio e + o —. Se il primo carattere non bianco di X$ non è uno dei caratteri permessi, otteniamo 0.

Esercizio 5.8 *A è un intero. Trovare il numero delle sue cifre.*

Esercizio 5.9 *Stampare B sopprimendo 3 cifre decimali.*

RICAPITOLAZIONE

Siamo ormai giunti ad uno stato perfezionato del nostro programma di gioco.
Esplorandolo abbiamo scoperto le principali possibilità del Basic.
Le istruzioni descritte finora valgono in generale per tutti i computers.
Ora vedremo delle proprietà particolari del C64 che, per la maggior parte, non si ritrovano sugli altri personal e, in particolare, le possibilità grafiche e sonore.
Ma prima, trattiamo due esercizi fondamentali.

Esercizio 5.10 *Calcolare il punteggio medio di ogni giocatore SM(T). Stampare il nome e il punteggio del giocatore che ha il punteggio massimo. In caso di parità, si sceglie il primo trovato.*

È particolarmente necessario qui ragionare sull'ordinogramma per studiare il problema del massimo. Questo problema molto classico si pone in innumerevoli applicazioni.

Esercizio 5.11 *Stampare la classifica dei giocatori, e non soltanto il primo.*

Richiamo: soluzioni agli esercizi vengono proposte in appendice.

Capitolo 6

Programmi grafici; colore e suono

L'istruzione più semplice che permette di fare disegni sullo schermo, la conosciamo già. Si tratta semplicemente dell'istruzione PRINT.
PRINT "stringa di caratteri" fa stampare una stringa di caratteri sullo schermo. Ciò permette di stampare un testo, come abbiamo già visto. Ma il C64 possiede tutto un insieme di caratteri detti grafici, particolarmente adatti al disegno, che possono, come gli altri, venir incorporati tra gli apici di un'istruzione PRINT.
Per disegnare con istruzioni PRINT, useremo una caratteristica del C64 che si rivelerà molto potente: dopo aver fatto un disegno sullo schermo, batteremo all'inizio di ogni linea di disegno: numero ?'Return'. Abbiamo costruito un programma il cui effetto sarà di stampare il disegno considerato.
Ciò proviene dalle proprietà già viste del tasto 'Return' per la scrittura di un programma.
Questo modo di procedere è molto pratico. Infatti, l'"artista" può mettere a punto il proprio disegno, correggerlo, spostarlo (a destra con i tasti 'Shift' 'Inst/Del', e verso l'alto tutto in un colpo portando il cursore sull'ultima linea dello schermo e battendo 'Return').
Facciamolo. A titolo dimostrativo disegneremo una casa.
Prima di tutto cancellare lo schermo con 'Clr'.
Portare il cursore al centro dello schermo, un po' sulla destra poi battere / ('Shift' N), —('C=' T) 7 volte e \ ('Shift' M). È la prima linea del disegno. 'C=' rappresenta il tasto 'Commodore'. Con movimento del cursore (e non con il tasto 'Return' che farebbe stampare SYNTAX ERROR) portare il cursore sulla linea successiva, uno spazio più a sinistra del / della linea precedente. Battere / ('Shift' N), 9 spazi e \ ('Shift' M). Il tetto si profila.

Linea successiva: sotto il / con movimenti del cursore. Battere: —('C=' T), ┌ ('Shift' 0), —('C=' T) 7 volte, ┐ ('Shift' P) e —('C=' T).

Linea successiva: spazio | ('C=' G), / ('Shift' N, —('C=' T), / ('Shift' M), spazio, ┌ ('C=' A), ┬ ('C=' R), ┐ ('C=' S) e | ('C=' M).

Linea successiva: spazio, | ('C=' G) 2 volte, spazio, | ('C=' M), spazio, ├ ('C=' Q), + ('Shift' +), ┤ ('C=' W), | ('C=' M).

Linea successiva: spazio, | ('C=' G) 2 volte, spazio, | ('C=' M), spazio, └ ('C=' Z), ┴ ('C=' E), ┘ ('C=' X), | ('C=' M).

Ultima linea: —('C=' @), └ ('Shift' L) 2 volte, —('C=' @), ┘ ('Shift' @), —('C=' @) 4 volte, ┘ ('Shift' @) e —('C=' @).

Ecco fatto! Otteniamo una casa che ha l'aspetto rappresentato qui di fronte. Oh! manca il camino! Nessun problema per aggiungerlo. Portate il cursore giusto sopra il tetto, sopra la finestra e battete ■ ('C=' I) e giusto sotto ■ ('C=' U).

Ora dovete aver preso dimestichezza con una buona parte dei caratteri grafici del C64. Provate altri disegni, ispirati da questo, o completamente diversi. Ma prima dobbiamo salvare il nostro disegno di casa. Per questo, bisogna trasformarlo in programma. È facile: all'inizio di ogni linea, battere 10?", poi 15?" ecc. fino a 50.
Facendo LIST dobbiamo vedere apparire il programma C.1 (ogni linea è in realtà listata su due linee, sullo schermo):

PROGRAMMA C.1

Ora, eseguite il programma mediante l'ordine RUN: la casa apparirà sullo schermo, sbarazzata naturalmente dei numeri di linea e delle istruzioni!

Esercizio 6.1 *Aggiungete un filo di fumo sopra il camino.*

Domande: le istruzioni del programma C.1 hanno degli apici non chiusi. *Non fa niente, il Basic del C64 ne genera uno fittizio alla fine della linea.*

Non ottengo la casa ma lettere miste a segni grafici. *È perché state in modo testo. Premete simultaneamente* 'Shift' *e* 'Commodore'.

Due importanti raccomandazioni per ottenere i risultati migliori:

— si devono usare numeri di istruzioni della stessa lunghezza, altrimenti vi saranno spostamenti;
— si deve mettere il disegno che si prepara abbastanza sulla destra dello schermo, perché, dopo, sarà spostato verso sinistra.

Un'ultima nota: questo metodo non permette di incorporare caratteri a contrasto invertito nel disegno.

Caratteri movimento del cursore

Il programma C.1 presenta diverse imperfezioni. La prima è che il disegno appare in un posto qualsiasi dello schermo e che non è solo: ciò che avevamo sullo schermo prima di fare RUN appare in alto dello schermo.
In modo diretto, è facile rimediarvi: possiamo svuotare lo schermo con il tasto 'Shift' 'Clear/Home' e portare il cursore dove vogliamo con i tasti movimento del cursore.
Ebbene, tutto ciò è pure possibile in modo programmato. Il segreto? Semplicissimo: includendo i caratteri della visualizzazione (chiamiamo in questo modo i caratteri come 'Clr', movimenti del cursore,- 'Rvs' ecc.) nella stringa di caratteri tra apici; diverranno effettivi soltanto al momento dell'esecuzione. Ad esempio, aggiungete al programma C.1 l'istruzione: 5 ?"Clr".
Lo schermo non viene svuotato quando battiamo il 'Clr' inserendo l'istruzione. Lo sarà quando avremo fatto RUN.
Quando abbiamo battuto l'istruzione, è avvenuto qualcosa di strano: battendo 'Clr' lo schermo non si è svuotato, ma è stato stampato un carattere nella stringa. Infatti, il C64 deve pur mostrare qualche cosa. Esiste una corrispondenza tra i caratteri di comando e la loro rappresentazione visibile, che diamo qui sotto. Ogniqualvolta ci troviamo tra apici e battiamo un carattere comando di visualizzazione, è la rappresentazione visibile quella che appare, poiché il C64 differisce l'esecuzione del comando fino all'esecuzione dell'istruzione.
I caratteri della rappresentazione sono tutti in contrasto invertito, per essere più riconoscibili, ma si pone una domanda: *si possono stampare quei caratteri come disegno?*: sì, verranno usati altri tasti a questo scopo.
Ma ci serve, in questo libro, una rappresentazione dei caratteri di comando, per darvi istruzioni da battere.
Seguiremo la convenzione seguente: il nome inizia con una maiuscola seguita da minuscola, il nome è sottolineato se non c'è 'Shift' e sopralineato se c'è 'Shift'. Esempio 'Home' si distingue da HOME (4 carat-

teri) 'Clr' è premuto con 'Shift'. Abbiamo già usato questa convenzione per designare i tasti 'Return' o 'Space'. Infine, per i caratteri ottenuti con il tasto 'Ctrl', scriviamo xxx^c, Rvs^c e Off^c. Diremo che questa è la rappresentazione libro.

Tab. 6.1. **I caratteri di comando di visualizzazione.**

Rappresentazione libro	Rappresentazione visibile C64	Effetto
'Shift' o ‾‾‾	...	Carattere dell'alto del tasto
'Espace' o Sp	□	Spazio
'Return'	...	Ritorno carrello
d	[■]	Cursore a destra
g	[■]	Cursore a sinistra
s	[Q]	Cursore in giù
a	[●]	Cursore in su
Home	[S]	Cursore all'origine (in alto a sinistra)
Clear o Clr	[♥]	Svuotare lo schermo
Rvs^c	[R]	Inversione di contrasto (nero su bianco)
Rvs Off^c	[–]	Ritorno al contrasto normale (bianco su nero)
Del	[T]	Soppressione di carattere
Inst	[I]	Inserzione di carattere

NB. La tabella 6.1 verrà completata nel seguito di questo capitolo con i caratteri di comando di colore.

Esercizio 6.2 *Quale stampa produce l'istruzione:*

10 PRINT "AAAss"; "Inst Inst BB"

Gli unici caratteri che non si possono includere in una stringa sono l'apice (") — che può concludere la stringa — e il 'Return'. Ma si possono stampare degli apici facendo:

?CHR$(34) e si può simulare un ritorno carrello con ?CHR$(13)

Siamo ora in grado di far stampare la nostra casa in mezzo allo schermo, cominciando il programma con Clr seguito da dieci cursore —in—giù. Terminiamo con 60 GOTO 60 per evitare la stampa del READY. Potete aggiungere spazi nei PRINT affinché la casa sia più a destra.
Ed ecco il programma C.2:

PROGRAMMA C.2

Domanda: con 60, ho un programma che non termina mai. Come uscirne? *Si preme il tasto* 'Stop'.

Stampa dell'ora

Invece di ciclare sul GOTO 60, si potrebbe stampare l'ora. Questo presuppone, ovviamente, di aver regolato l'ora in modo diretto: TI$ = "123400" (se sono le 12 e 34). Non si dimentichi che TI$ è una stringa di 6 caratteri HHMMSS.
Ma un semplice 60 ?TI$: GOTO 60 non è adeguato: copre lo schermo di messaggi diversi.
Ciò di cui abbiamo bisogno, è di stampare il TI$ sempre nello stesso posto. I movimenti del cursore ci forniscono la soluzione. Dopo una stampa, siamo pronti a stampare immediatamente sotto. Bastano quindi una '*ā*' per risalire di una linea, e 18 '*d*' perché l'ora sia sotto la casa. L'istruzione 52 PRINT genera una linea intermedia ed evita che l'ora sia attaccata alla casa.
Otteniamo allora il programma C.3A.

PROGRAMMA C.3A

Esercizio 6.3 *È più bello ancora quando l'ora viene stampata a contrasto invertito.*

Esercizio 6.4 *Vorremmo stampare l'ora, ma nella forma hhHm mMssS.*

Per questo, si usano le funzioni stringhe di caratteri: il numero delle ore è LEFT$(TI$,2), quello dei secondi RIGHT$(TI$,2) e quello dei minuti è MID$(TI$,3,2). Dal che segue la nuova fine del programma (C.3B) dopo 50:

```
52 PRINT
55 H$ = LEFT$(TI$,2): M$ = MID$(TI$,3,2): S$ = RIGHT$(TI$,2)
60 PRINT "äddddRvs"H$"SpHSp"M$"SpMSp"S$"SpSSp": GOTO 55
```

Si noti che i ";" sono inutili, ad esempio, tra H$ e "SpHSp".

Scritta sullo schermo da *POKE*

Esiste un altro modo di scrivere l'ora in un posto fisso dello schermo. Questo modo deriva dalle due possibilità di accesso diretto alla memoria offerte dal C64, le istruzioni *PEEK* e *POKE*.

PEEK permette di leggere dove si vuole in memoria

PEEK (x) fornisce il valore compreso fra 0 e 255 del byte d'indirizzo x (in decimali).
Ad esempio ?PEEK(0) fa stampare 47 poiché all'indirizzo 0 della memoria, c'è 47.
Se la prima istruzione del vostro programma è 5, allora ?PEEK(2051) fa stampare 5. Potete in tal modo venire a conoscenza del contenuto di qualsiasi locazione della memoria.

POKE permette di scrivere in una locazione della memoria

POKE x, y scrive nell'indirizzo x il valore y (y deve essere compreso tra 0 e 255, x tra 0 e 65535).
Ad esempio, POKE 1024.0 scrive il numero 0 nella casella di memoria di indirizzo 1024. Fatelo.
Toh, è apparso un segno @ in alto a sinistra dello schermo. La ragione è semplice: una zona della memoria serve a rinfrescare lo schermo. Ciascuna locazione di tale zona contiene il codice-carattere del carattere che deve essere visualizzato nella posizione corrispondente dello schermo. Siccome vi sono 25 linee di 40 colonne sullo schermo, ossia 1000 posizioni, la zona comprende 1000 bytes. Va da 1024 a 2023, in conformità allo schema qui sotto.

```
(1024)(1025)... (1063)
(1064)(1065)... (1103)
 .
 .
 .
(1984)......... (2023)
```

La posizione di stampa dello schermo linea I colonna J corrisponde all'indirizzo:
1023 + (I — 1) ∗ 40 + J

Ma non è così semplice. È possibile che quando avete fatto POKE 1024, 0 non abbiate fatto apparire alcuna @. Ciò può essere dovuto al fatto che, essendo in basso allo schermo al momento del POKE, la @ è subito stata cancellata al momento dello spostamento in alto dello schermo per avere READY.
Per eliminare questa causa, fate 'Stop' 'Restore' lo schermo si svuota. Poi fate POKE 1024, 0 dopo essere risaliti alla prima linea dello schermo. Allora ottenete la stampa in alto dello schermo:

```
@ OKE 1024, 0
READY
—
—
```

La @ è effettivamente apparsa. Proviamo quindi di farne apparire una in basso a destra dello schermo con:

POKE 2023, 0

Non accade nulla. In realtà, la @ è effettivamente apparsa. Ma, è stata scritta in azzurro su sfondo azzurro, dunque non la vedete! per verificarlo, portate il cursore in basso, a destra dello schermo (attenti, non superate la posizione, altrimenti tutto risalirà). Vedete in azzurro la vostra @.
In realtà, c'è una seconda memoria associata allo schermo: occupa, pure lei, 1000 posizioni da 55296 a 56295, e all'indirizzo 55295

$+(I-1)*40+J$ si trova il codice del colore di stampa del carattere in linea I colonna J dello schermo.
Per verificare, riportate il cursore verso il centro dello schermo e fate POKE 56295,2. La @ appare effettivamente in nero sullo schermo.
I differenti codici colore verranno descritti nella sezione del presente capitolo che tratta della questione dei colori.

Esercizio 6.5 *Visualizzare una A verde sulla 3ª linea 4ª colonna dello schermo (il codice-schermo di A è 1, il codice colore del rosso è 5).*

Per stampare un carattere, disponiamo ora di tre metodi:

— o ?"carattere";
— o ?CHR$(ascii);
— oppure POKE posizione, codice-schermo.

I codici ASCII ed i codici schermo sono riassunti nella tabella a pagina seguente.

Annotazioni sulla tabella 6.2

+ : quando un codice ASCII, a, è seguito da +, è perché si può usare equivalentemente il codice $a+64$;
* : per ottenere il carattere in contrasto invertito, aggiungere 128 al codice schermo;
— : per i tasti lettere se codice ASCII del carattere $x=a$ allora codice ASCII DI Shift $x=a+128$.

Rimane da spiegare perché certe linee della tabella contengono due caratteri. Ciò è dovuto al fatto che la visualizzazione funziona in due modi. Il carattere di destra (nella tabella) appare quando si è in modo grafico. Il cattere di destra (nella tabella) appare quando si è in modo testo. All'accensione e dopo 'Stop' 'Restore' si è in modo grafico. Si passa da un modo all'altro premendo simultaneamente 'Commodore' e 'Shift'. Ad esempio, fate ?CHR$(65) CHR$(216). Ottenete una A maiuscola e un trifoglio. Se premete 'C=' e 'Shift', la A maiuscola diventa a minuscola e il trifoglio diventa una X maiuscola.
Sono i caratteri grafici che stanno sulla destra dei tasti (ottenuti con SHIFT) quelli che, in modo testo, vengono sostituiti dalle lettere maiuscole. I caratteri grafici ottenuti con 'Commodore' rimangono. L'unica eccezione è il tasto *: ⊟ (ottenuto con 'Shift') rimane mentre ◥ diventa ▧ .
Si noti che, anche in modo testo, si dispone di un certo numero di caratteri grafici: tutti quelli che, nella tabella 6.2, stanno da soli in una colonna. In particolare, i caratteri che aiutano a presentare tabelle di numeri sono sempre disponibili.
Il nostro primo uso di ciò, sarà di scrivere l'ora in alto a sinistra del-

Tab. 6.2. **Codici caratteri del C64 (in decimali).**

Carattere		Codice schermo*	ASCII
A	a	1	65
B	b	2	66
C	c	3	67
D	d	4	68
E	e	5	69
F	f	6	70
G	g	7	71
H	h	8	72
I	i	9	73
J	j	10	74
K	k	11	75
L	l	12	76
M	m	13	77
N	n	14	78
O	o	15	79
P	p	16	80
Q	q	17	81
R	r	18	82
S	s	19	83
T	t	20	84
U	u	21	85
V	v	22	86
W	w	23	87
X	x	24	88
Y	y	25	89
Z	z	26	90
0		48	48+
1		49	49+
2		50	50+
3		51	51+
4		52	52+
5		53	53+
6		54	54+
7		55	55+
8		56	56+
9		57	57+
+		43	43+
-		45	45+
*		42	42+
/		47	47+
↑		30	94
.		46	46+
=		61	61+
<		60	60+
>		62	62+
!		33	33+
?		63	63+
:		58	58+
;		59	59+
,		44	44+

Carattere		Codice schermo*	ASCII
(		40	40+
)		41	41+
"		34	34+
'		39	39+
#		35	35+
$		36	36+
%		37	37+
&		38	38+
£		28	92
←		31	95
π		94	222
[		27	91
]		29	93
@		0	64
	A	65	193
	S	83	211
	Z	90	218
	X	88	216
		125	189+
		109	173+
		112	176+
		110	174+
		126	190+
		124	188+
		108	172+
		123	187+
Sp		32/96	32+
	O	79	207
	P	80	208
		122	186+
	L	76	204
	N	78	206
	M	77	205
	V	86	214
		91	219
		127	191
	W	87	215
	Q	81	209
		116	180+
		117	181+
		97	161+
		118	182+
		106	170+
		105	169+
		95	223

Carattere		Codice schermo*	ASCII
		102	166
		104	168+
		92	220
		119	183+
		120	184+
		98	162+
		121	185+
		111	175+
		114	178+
		115	179+
		107	171+
		113	127+
	J	74	202
	K	75	203
	U	85	213
	I	73	201
		99	163+
	E	69	197
	D	68	196
	C	67	195
		64	192
	F	70	198
	R	82	210
		100	164+
		101	165+
	T	84	212
	G	71	199
	B	66	194
		93	221
	H	72	200
	Y	89	217
		103	167+
Return			13+
d			29
s			157
b			17
a			145
Home			19
Clr			147
Del			20
Inst			148
Rvs[C]			18
Off[C]			146

lo schermo. Per questo, piazzeremo ciascuno dei 6 caratteri di TI$ agli indirizzi 1024 e successivi e vi imporremo il colore nero (codice 0):

```
200 FOR I = 1 TO 6 : POKE 1023 + I, ASC(MID$(TI$, I))
210 POKE 55295 + I, Ø : NEXT I: GOTO 200
```

Ricordatevi il funzionamento delle funzioni di stringhe di caratteri: MID$(TI$, I) prende i caratteri di TI$ dall'Iesimo in poi, ASC prende il codice ASCII del primo di essi.
Sfruttiamo in questo caso il fatto che TI$ è formato da cifre per le quali (tab. 6.2) il codice schermo è uguale al codice ASCII.

Esercizio 6.6 *Scrivere l'ora in mezzo allo schermo. L'ora deve essere da sola sullo schermo ed apparire in contrasto invertito.*

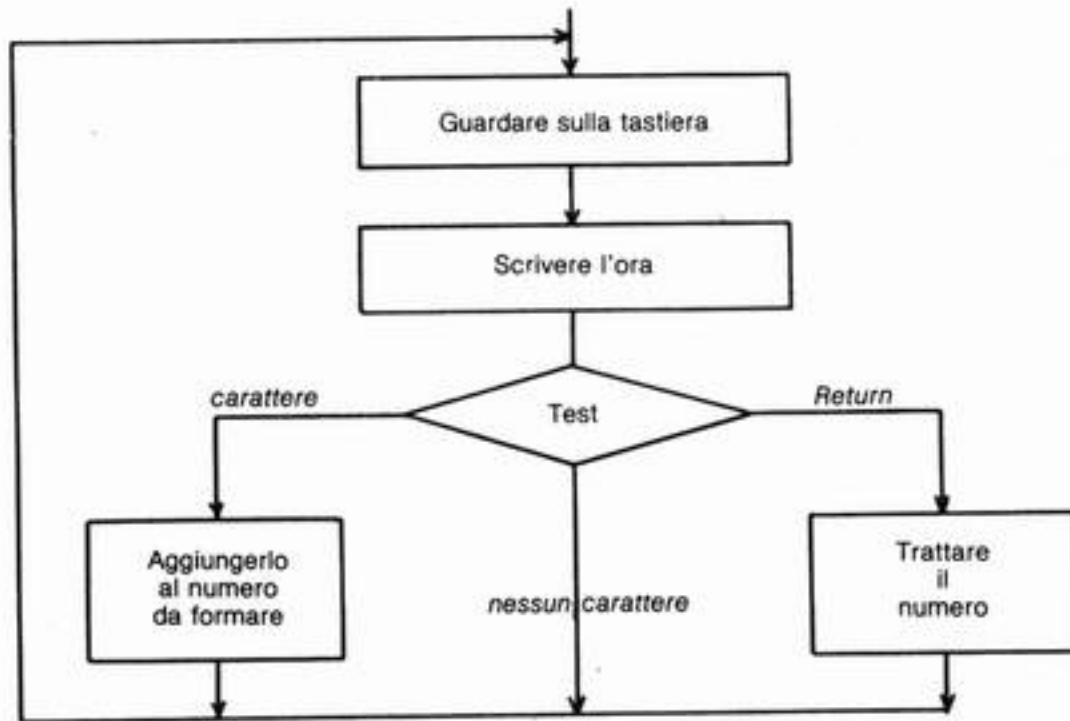
L'istruzione *GET*

Adesso che sappiamo scrivere l'ora in un posto fisso dello schermo, possiamo apportare un miglioramento spettacolare al nostro programma di gioco del capitolo precedente.
Quale angoscia per il giocatoe nel vedere i secondi susseguirsi sullo schermo mentre cerca il numero da indovinare!
Per vedere meglio l'essenziale, partiamo da una versione semplificata del programma B.5/B.8:

PROGRAMMA B.5B

```
10 PRINT"♥":C=1+99*RND(1):TI$="000000"
20 IF TI>3600 GOTO 70
30 PRINT"NM.PROPOSTO";
35 FOR I=3 TO 6:POKE1056+I,ASC(MID$(TI$,I)):POKE55328+I,
                                                        0:NEXT
40 INPUT A
50 E=100*ABS(A-C)/C:IF E<1 GOTO 80
60 PRINT"ERRORE";INT(E);"%":GOTO 20
70 PRINT"PERSO!" :GOTO 90
80 PRINT"VINTO!"
90 INPUT"RICOMINCIAMO";A$:IF A$="SI" GOTO 10
```

Le semplificazioni che abbiamo portato sono ovvie: abbiamo soppresso qualsiasi gestione dei vari giocatori, semplificando il trattamento degli errori, e portato al minuto il tempo limite.
La fine del programma è degna di attenzione: si chiede al giocatore se desidera ricominciare e si analizza la stringa di caratteri formata dalla sua risposta; è una cosa molto usata in tutti i programmi detti "interattivi".

Abbiamo messo il messaggio "NM.PROPOSTO" (linea 30) in un'istruzione PRINT distinta da INPUT. Infatti è in mezzo ad esse che si collocherà la stampa dell'ora. Si scrivono nella parte destra della prima linea dello schermo le cifre dei minuti e dei secondi di TI$ mediante l'istruzione 35.
Proviamo il programma ottenuto in tal modo.
Non funziona come vogliamo: non vediamo scorrere il tempo mentre il giocatore esita a dare il suo numero.
Tutto proviene dall'istruzione INPUT. Quando un'istruzione INPUT si esegue, il programma rimane bloccato fino a quando si preme il tasto 'Return'. Intanto, nulla può essere fatto.
Quello che ci vorrebbe, è un'istruzione che guardi se abbiamo battuto un carattere alla tastiera e che ridia il controllo all'utente in ogni caso. Si potrebbe quindi seguire il seguente schema a blocchi:

Ebbene il C64 dispone di una tale istruzione, GET. *GET B$ fornisce in B$: o il carattere sul quale abbiamo premuto, oppure la stringa vuota se non abbiamo premuto alcun carattere:* proprio quello che ci vuole. Applicando esattamente il precedente schema a blocchi, otteniamo:

PROGRAMMA B.5C

```
10 PRINT"♫":C=1+99*RND(1):TI$="000000"
20 PRINT"NM.PROPOSTO?  ";
25 A$=""
30 IF TI>3600 GOTO 70
```

```
33 GET B$
35 FOR I=3 TO 6:POKE1056+I,ASC(MID$(TI$,I)):POKE55328+I,
                                                     0:NEXT
37 IF B$="" GOTO 30
39 IF B$=CHR$(13) GOTO 45
41 A$=A$+B$:PRINT B$;:GOTO 30
45 A=VAL(A$):PRINT
50 E=100*ABS(A-C)/C:IF E<1 GOTO 80
60 PRINT"ERRORE";INT(E);"%":GOTO 20
70 PRINT"PERSO!" :GOTO 90
80 PRINT"VINTO!"
90 INPUT"RICOMINCIAMO";A$:IF A$="SI" GOTO 10
```

Ci limiteremo ora ai commenti essenziali. Il cambio dell'ordine di 20 e 30 è essenziale affinché si passi ad ogni momento sul test del tempo, di modo da essere ben sicuri di non lasciare più di un minuto. Notiamo che, in 20, viene stampato esplicitamente il punto interrogativo: GET non lo dà, all'inverso di INPUT. Parimenti con GET, siamo costretti a controllare l'impaginazione, ciò giustifica gli spazi in 20 e 60, e siamo costretti a stampare i caratteri via via che li troviamo, e questo giustifica il PRINT in 41.
Le linee 25, 33 e 41 costruiscono progressivamente la stringa A$ partendo da un valore iniziale "stringa vuota". Ogni qualvolta è disponibile un carattere B$, esso viene concatenato con la stringa A$ già ottenuta: A$ = A$ + B$. Si noti pure come, in 36, venga testato il carattere 'Return'; non si poteva fare: 39 IF B$ = "Return", poiché il "Return" termina spietatamente la linea. Potevamo anche scrivere:

39 IF ASC(B$) = 13 . . .

Consigliamo vivamente di adattare quest'ultima versione di B.5 a B.10 che avevamo ottenuto nel capitolo precedente; praticamente, cambiamo soltanto i numeri d'istruzione.
Vediamo che, in pratica, l'istruzione GET "coglie" i caratteri uno per uno sulla tastiera. Questo permette una migliore interattività tra l'utente e la macchina. Ad esempio, alla fine di B.5C, potremmo incominciare non appena l'utente ha battuto S di SI, uscire non appena batte N, invece di aspettare SI "Return". Questo permette di reagire più in fretta a ciò che l'utente batte: è necessario, soprattutto quando, ad esempio in un gioco di battaglia di carri armati, la pressione di un tasto simula un tiro.

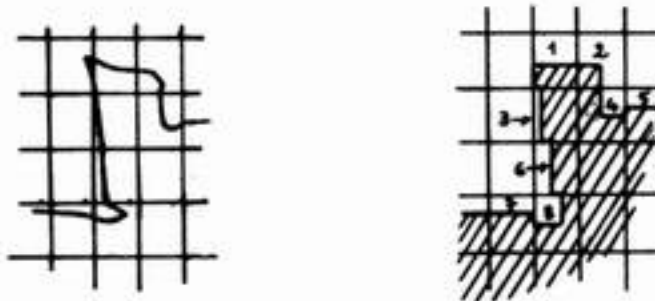
Esercizio 6.7 *Fate la modifica necessaria alla fine del programma B.5C.*

Esercizio 6.8 *È spesso utile inserire in un programma un punto d'attesa: ciò permette di fermare un messaggio affinché l'utente ne prenda*

conoscenza. Molto spesso, si decide che basta, per continuare, che l'utente prema qualsiasi tasto. Create la sequenza che attua questo comportamento.

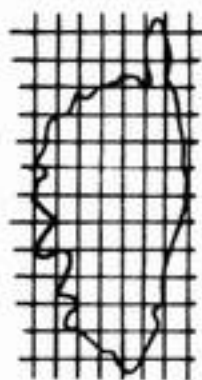
Esercizio 6.9 *Il programma C.3B somiglia parecchio al comportamento di un orologio a cucù. Affinché la somiglianza sia completa, basta che alle ore x 0mn 0s un cucù appaia alla porta. Fatelo. Fate lampeggiare il cucù mentre appare.*

Carta di Francia

Per mostrare la nostra padronanza dei grafici, disegneremo un profilo della Francia.
Per tale disegno, non è possibile procedere come per la casa, cioè preparando il tracciato sullo schermo e mettendo i PRINT. Infatti, alcuni caratteri verranno fatti in reverse (ad esempio non esiste ◨ : lo si ottiene per inversione di ◧ , ossia 'Rvs' 'Shift'!). Ora, i caratteri invertiti si ottengono soltanto usando i 'Rvs' programmati, esplicitamente messi tra apici.
L'unico metodo è il seguente: sovrapporre una carta di Francia e una quadrettatura geometricamente simile alla "maglia" dello schermo. Usate una griglia di 4 mm di larghezza su 5 mm di altezza. Poi, cercate, riga per riga, i caratteri che più si avvicinano alle curve da riprodurre.

Si vedono, qui sopra, il Cotentin (Normandia), e l'approssimazione scelta (in scala 2), dal che seguono i caratteri nei quadratini:

1. 'C=' I
2. 'C=' F
3. Rvs^c 'C=' G
4. Rvs^c 'C=' C
5. Rvs^c 'C=' U
6. Rvs^c 'C=' J
7. Rvs^c 'C=' U
8. Rvs^c 'C=' V

Ovviamente, i pieni si fanno con Rvs^c Sp.
Ne segue il programma C.5. La linea 1040 serve ad impedire la stampa di READY. L'abbiamo già usata.

```
1010 PRINT"
1011 PRINT"
1012 PRINT"
1013 PRINT"
1014 PRINT"
1015 PRINT"
1016 PRINT"
1017 PRINT"
1018 PRINT"
1019 PRINT"
1020 PRINT"
1021 PRINT"
1022 PRINT"
1023 PRINT"
1024 PRINT"
1025 PRINT"
1026 PRINT"
1027 PRINT"
1028 PRINT"
1029 PRINT"
1030 PRINT"
1031 PRINT"
1032 PRINT"
1033 PRINT"
1034 PRINT"
1040 GOTO1040
```

Chiediamo ai lettori Corsi di volerci scusare per non aver fatto figurare la loro bella isola sulla nostra carta. Per risarcirli, proponiamo l'esercizio seguente:

Esercizio 6.10 *Cartografate la Corsica con l'aiuto della griglia qui di fronte.*

Esercizio 6.10 bis *Fate allo stesso modo una carta dell'Italia, o della Svizzera, o del Canada.*

Dobbiamo notare che i risultati sono abbastanza mediocri: al momento siamo in "grafica bassa risoluzione"; l'alta risoluzione (prossimo capitolo) ci permetterà di migliorare parecchio.

Quiz geografici - I sottoprogrammi

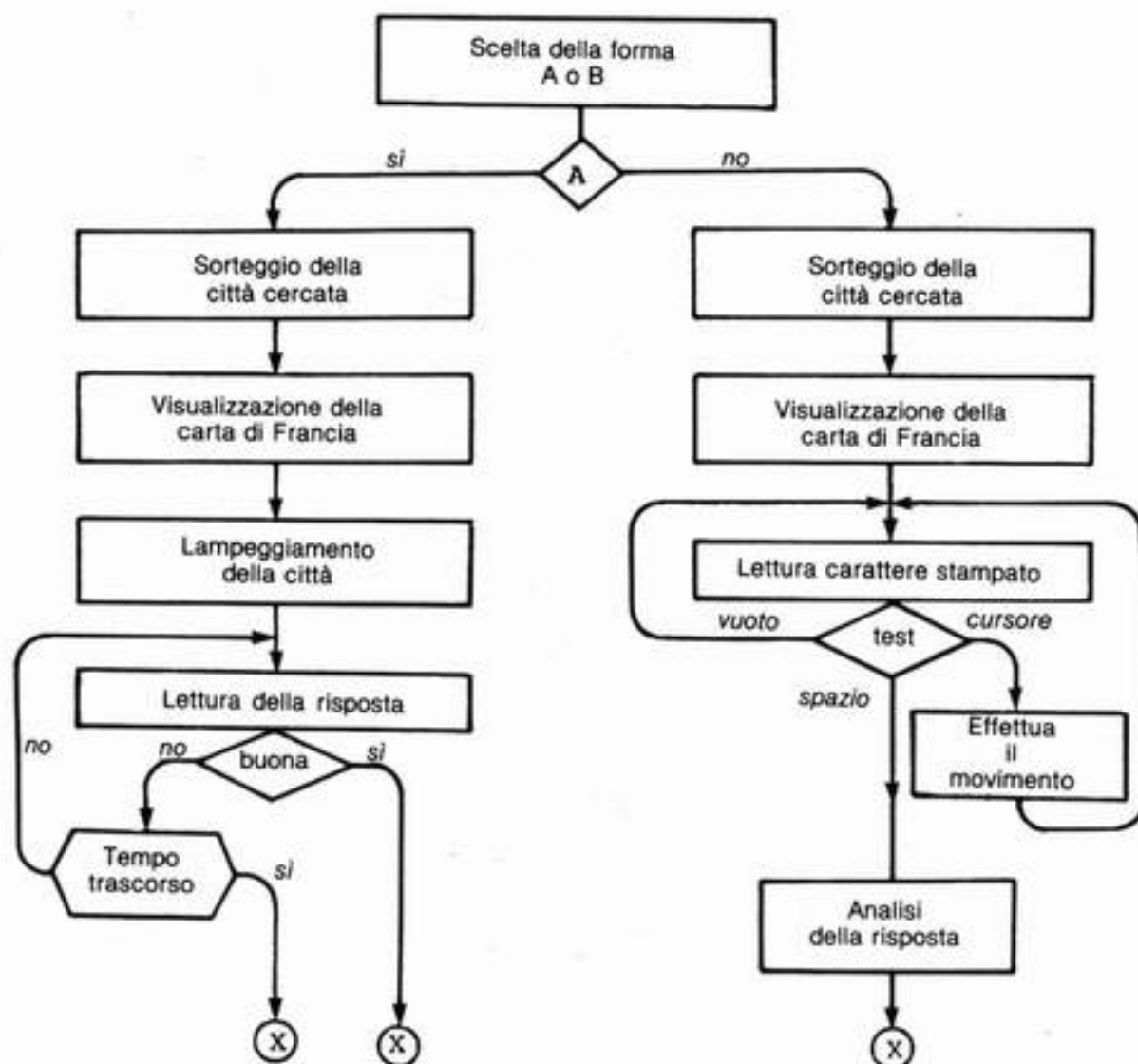
Adesso che disponiamo di una carta di Francia, potremmo usarla per giocare ad un gioco utile. Ci permetterà di rivedere le nostre conoscenze geografiche. Si tratta di riconoscere la posizione delle principali città francesi. Si può giocare in due modi, e li attueremo entrambi.

A. Il programma fa lampeggiare un quadratino sullo schermo; avete 20 secondi per battere il nome della città che vi si trova.

B. Il programma dà un nome di città. Dovete, con l'aiuto dei tasti movimento del cursore, portare il carattere ▒ ('C = ' +) sulla posizione di tale città.

Battete Return quando siete soddisfatti della posizione raggiunta. Ovviamente, disponete di un tempo limitato. Questa è la base dei programmi d'insegnamento guidato da computer, ormai diffusissimi. Naturalmente, li si possono perfezionare esaminando gli errori commessi, visualizzando indicazioni per aiutarvi e tenendo il conto dei risultati.

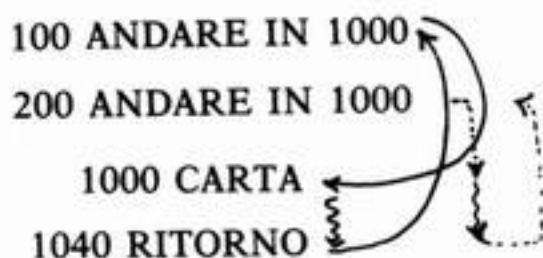
Abbiamo visto in precedenza tutti gli elementi che permettono di farlo; ora ci limiteremo alle soluzioni più semplici.

Il programma segue lo schema a blocchi generale qui sotto:

Questo schema è semplificato. Quello che appare fondamentalmente è che, in due posti, bisogna eseguire la stessa operazione: visualizzare la carta di Francia. Il programma della carta è di parecchio troppo lungo per ripeterlo in due posti. Andrebbe bene disporre del meccanismo seguente (cfr. figura qui sotto).

```
100 ANDARE IN 1000

200 ANDARE IN 1000

    1000 CARTA

 1040 RITORNO
```

Il programma da ripetere viene scritto una volta sola, dalla linea 1000 ad esempio.
Ogni qualvolta dobbiamo eseguirlo (qui in 100 e 200) una istruzione simile ad un GOTO fa saltare in 1000, dove viene eseguita la sequenza.
La sequenza termina con una istruzione che vuol dire "Tornate da dove venite". Essendo venuto dalla linea 100, tornare subito sotto 100 (percorso in tratto pieno). Essendo venuto da 200, tornare subito sotto 200 (percorso punteggiato).
In pratica, si tratta di un meccanismo di salto che si ricorda da dove viene. Questo meccanismo esiste in Basic.
Il programma usato più volte si chiama sottoprogramma. L'istruzione di salto verso il sottoprogramma è l'istruzione di richiamo GOSUB: 100 GOSUB 1000.
L'istruzione di ritorno si scrive semplicemente RETURN (le lettere RETURN, non il tasto Return').
Il programma ha quindi la struttura:

```
:
100 GOSUB 1000  }
:               } programma principale
200 GOSUB 1000  }
:
1000            }
     :          } sottoprogramma (qui il tracciato della carta
1040 RETURN     } di Francia)
```

Questa struttura è estremamente importante e potente. Un programma può richiamare diversi sottoprogrammi. Ogni sottoprogramma può richiamarne altri. In ogni caso, il C64 vi si raccapezzerà ed effettuerà i ritorni.
Una volta acquisita questa semplificazione del nostro lavoro, ci serve una lista di città con il loro nome e, per ciascuna di esse le sue coordinate. I (da 1 a 25) e J (da 1 a 40) che la localizzano sullo schermo. Naturalmente, tali coordinate saranno approssimative, così come il disegno della carta.
Ci limiteremo in questo caso alle 11 seguenti città (ciò evita le istruzioni DIM, e nulla vieta di aggiungerne ulteriormente).

Città	I	J	N	Città	I	J	N
Parigi	7	22	0	Nantes	11	12	6
Marsiglia	22	29	1	Strasburgo	7	34	7
Lione	15	27	2	St-Etienne	16	26	8
Tolosa	21	20	3	Le Havre	4	17	9
Nizza	20	34	4	Lille	2	23	10
Bordeaux	17	15	5				

I preliminari del programma sono quindi semplici. La tabella precedente viene messa sotto forma di DATA (linee 10, 15, 20) che vengono poi letti (linee 30 e 40). Poi, viene la scelta del modo (linee 50 e 60) (per un uso reale del programma, si dovrebbero stampare più spiegazioni di quello che facciamo).
Prima di battere questo programma, caricate il vostro programma carta di Francia e aggiungete una linea 1000 REM con, eventualmente, una spiegazione del sottoprogramma. Cambiate la linea 1040 in 1040 RETURN.

PROGRAMMA C.6. QUIZ GEOGRAFICO

```
10 DATA PARIGI,MARSIGLIA,LIONE,TOLOSA,NIZZA,BORDEAUX,NANTES,
                                                   STRASBURGO
15 DATA "SAINT ETIENNE","LE HAVRE",LILLE
20 DATA 7,22,22,29,15,27,21,20,20,34,17,15,11,13,7,34,16,26,4,
                                                    17,2,23
30 FORV=0TO10:READ NOM$(V):NEXT
40 FORV=0TO10:READ II(V),JJ(V):NEXT
50 INPUT"A O B";X$
55 V=INT(11*RND(1)):T=TI
60 IF X$<>"A" GOTO 500
```

In realtà, il sorteggio della città e l'inizializzazione del tempo sono comuni e andranno in 55:

$$55\ V = INT(11 * RND(1)) : T = TI$$

Gioco *A*

Ci troviamo dopo 70:

```
70 PRINT"GIOCO A:BATTETE IL NOME DELLA CITTA CHE LAMPEGGIA
                                                 POI RETURN"
72 FORI=1TO1000:NEXT
75 IJ=1023+40*(II(V)-1)+JJ(V)
80 GOSUB1000:A$="":POKEIJ,42:K=0
85 IFTI-T>1200GOTO170
90 P=PEEK(IJ):Q=P+128*SGN(128-P):POKEIJ,Q
95 GET B$:IFB$="" GOTO 85
```

```
100 IFB$=CHR$(13) GOTO150
105 K=K+1:L=ASC(B$)-64:IFL<0THEN L=L+64
110 POKE1623+K,L:POKE55895+K,14:A$=A$+B$:GOTO 75
150 IF A$=NOM$(V) GOTO 180
160 PRINT"]NO":FORI=1TO500:NEXT:GOTO 75
170 PRINT"]PERSO":GOTO190
180 PRINT"]VINTO"
190 INPUT"▮▮▮RICOMINCIAMO";A$
195 IF A$="SI" GOTO50
200 END
```

Il programma è costruito con materiale già visto, particolarmente la costruzione carattere per carattere del nome A$ della città proposta dall'alunno (istruzione 80 a 110).
In 72 e in 160, pause per fermare il messaggio visualizzato.
In 75, IJ è l'indirizzo sullo schermo della città cercata.
In 80, il 42 è il codice della stella che stampiamo alternativamente in normale o in reverse perché lampeggi.
In 90, facciamo lampeggiare la stella: P è il carattere attuale. Q è $P+128$ se $P>128$ e $P-128$ se $P<128$ (mediante la funzione segno). Quindi Q è comunque l'inverso di P.
In 105-110, stampiamo il carattere trovato; ma è meno semplice che con le cifre: dobbiamo togliere 64 per ottenere il codice schermo, tranne che per il carattere spazio che abbiamo già in due città. L'indirizzo di inizio 1623 assicura che il nome venga battuto in mezzo al Golfo di Guascogna! Il POKE 55895... assicura che la scrittura avvenga in azzurro e non in blu scuro su blu scuro. Il resto del programma non dovrebbe creare difficoltà; passiamo quindi al gioco B.

Gioco *B*

Questa volta siamo dopo la linea 500

```
500 PRINT"GIOCO B:CON I TASTI CURSORE,PORTATE ▮ SULLA CITTA
505 PRINT"▮▮▮▮▮▮▮▮▮";NOM$(V):FORI=1TO2000:NEXT
510 I=1:J=1
515 GOSUB 1000
520 IJ=1023+40*(I-1)+J:A=PEEK(IJ):POKEIJ,102:POKEIJ+54272,14
525 IFTI-T>7200GOTO170
530 GETA$:IFA$=""GOTO525
535 IFA$=CHR$(13)GOTO570
540 IFA$="▮"THENI=I+1
545 IFA$="]"THENI=I-1
550 IFA$="▮"THENJ=J+1
555 IFA$="▮"THENJ=J-1
560 POKEIJ,A:GOTO520
570 IF I=II(V) AND J=JJ(V) GOTO 180
575 PRINT"]NO":FORR=1TO500:NEXT:GOTO 515
```

I e J inizializzati ad 1, indicano la posizione del cursore grigio (codice schermo 102), indirizzo schermo IJ.
Di nuovo, per motivi di interattività, usiamo GET per l'esame dei tasti premuti. A seconda del tasto usato, il movimento viene effettuato, cioè I o J vengono modificati. Il tasto 'Return' viene testato da CHR$ dato che non lo si può mettere tra apici. Il movimento, linea 560 e 520, viene effettuato rimettendo il vecchio contenuto A in IJ. Il nuovo IJ viene quindi calcolato, e il cursore grigio spostato in questa nuova posizione.
Quando l'utente batte '*Return*', la I e la J raggiunte sono la posizione proposta. Questa viene confrontata coi dati della tabella II e JJ. Si noti l'uso della variabile R per la sosta in 575 in modo da non modificare I usato altrove!
L'ordine delle istruzioni, in 510, 515 è tale che quando la risposta è NO un altro tentativo è possibile. Si riprende in tal caso dalla posizione proposta nel tentativo precedente, il che permette di raggiungere la meta con piccole correzioni successive.
Per l'insieme dei tentativi, il tempo limite è di 2 minuti (contro 20s nel gioco A).

RICAPITOLAZIONE

Questo programma, che è l'archetipo dei programmi usati nell'insegnamento guidato da computer, utilizza soltanto tecniche semplici che sono state introdotte progressivamente nel corso del gioco "indovina un numero" e per la programmazione del disegno di una casa.
L'unica tecnica introdotta qui — ed essa è molto importante — è quella dei ***sottoprogrammi****.*
Con un semplice GOSUB, il tracciato della carta di Francia viene effettuato ogni volta è necessario nel programma.
L'istruzione GET porta pure una buona interattività nel dialogo uomo-macchina, essenziale in questa applicazione.
Non proponiamo qui alcun esercizio su questo programma, costruito passo per passo, ma non incoraggeremo mai troppo il lettore ad esercitarsi ad apportarvi tutte le modifiche che giudichi utile provare.
Potete in particolare ispirarvi alle ultime versioni del programma B per gestire i "punteggi" ottenuti da vari alunni. Potete anche cambiare i tempi e aggiungere città (in tal caso, attenti a DIM...).

Nota:

Il sottoprogramma carta di Francia è stato scritto da 1000 in poi, in modo da permettere l'uso in diversi programmi. Immagazzinatelo su cassetta, e quando desiderate incorporarlo ad un programma da scrivere, caricatelo per primo. Poi scrivete il vostro programma da 0 a 999.

Questo modo di procedere, che ci evita la fastidiosa copiatura di una parte del programma, si può generalizzare. Ma attenti! Potete usarla una sola volta per programma.

Ora dobbiamo vedere alcuni aspetti particolarmente spettacolari del C64: il colore ed il suono.

IL COLORE

Nel corso di questo libro abbiamo già fatto qualche allusione ai comandi di colore. Ora si tratta di precisare tutte le possibilità offerte dal C64 in questo campo.
Così come le scritte sullo schermo possono essere fatte con dei PRINT o dei POKE, i comandi colore sono al livello elementare con PRINT e al livello elaborato con POKE. Alcuni comandi sono accessibili soltanto con POKE.

Comandi elementari

Il comando più elementare consiste nel dire "il prossimo carattere che stamperemo, voglio che sia, ad esempio, in **rosso**". Lo si fa molto semplicemente mediante i tasti di comandi di colore sulla lina di sopra della tastiera; il comando di colore si ottiene premendo simultaneamente il tasto 'Ctrl'.
Ad esempio se battete la sequenza:

'Ctrl' GRN XXX 'Ctrl' PUR YYY

otterrete tre X verdi e tre Y porpora.

NB. All'accensione i caratteri vengono stampati in azzurro (su blu scuro). I caratteri di controllo di colore possono essere messi tra apici per agire in modo differito al momento dell'esecuzione del programma. Anche in questo caso, vengono rappresentati con un motivo in contrasto invertito.
La tabella 6.3 completa le tabelle 6.1 e 6.2 con i caratteri di colore.

Esercizio 6.11 *Coprire lo schermo di stelle di colori casuali (bianco escluso).*

Lo sfondo e il quadro

Perché c'è un colore blu quando, se chiediamo di scrivere caratteri blu, non li vediamo sullo sfondo blu dello schermo? Ebbene, perché possiamo cambiare il colore dello sfondo. Ciò si fa con POKE ad un indirizzo che, in realtà, è uno dei registri del circuito integrato 6567 che gestisce tutto quello che viene visualizzato dal C64. Questo circui-

Tab. 6.3. **I caratteri di comando di colore.**

Rappresentazione del libro (^{c} vuol dire con '*Ctrl*')	**Rappresentazione visibile C64**	**Codice ASCII (CHR$())**	**Effetto**
Blk^c		144	Nero
Wht^c		5	Bianco
Red^c		28	Rosso
Cyn^c		159	Turchese
Pur^c		156	Porpora
Grn^c		30	Verde
Blu^c		31	Blu
Yel^c		158	Giallo
$_c$Blk		129	Arancione
$_c$Wht		149	Marrone
$_c$Red		150	Rosa
$_c$Cyn		151	Grigio scuro
$_c$Pur		152	Grigio medio
$_c$Grn		153	Verde chiaro
$_c$Blu		154	Azzurro
$_c$Yel		155	Grigio chiaro

to integrato si chiama VIC (Video Interface Chip = cassetta di interfaccia video). È tanto importante e sofisticato quanto il microprocessore.
C'è, nella cassetta VIC, un certo numero di registri il cui contenuto può essere cambiato da POKE. Il contenuto determina la gestione di quello che viene visualizzato; quindi un modesto POKE può modificare, in modo spettacolare, quello che appare sullo schermo. In questo libro non vedremo che i principali registri e i principali effetti che si possono ottenere.

I registri d'indirizzi rispettivi 53281 e 53280 determinano il colore dello sfondo dello schermo come pure quello del quadro (sul quale non viene mai scritto nessun carattere), secondo il codice in 4 bits:

Tab. 6.4. **Codice dei colori.**

Codice+	**Colore**	**Codice**	**Colore**
0	Nero	8	Arancione
1	Bianco	9	Marrone
2	Rosso	10	Rosa
3	Turchese	11	Grigio scuro
4	Porpora	12	Grigio medio
5	Verde	13	Verde chiaro*
6	Blu	14	Azzurro
7	Giallo	15	Grigio chiaro

+ Se il colore si ottiene con 'Ctrl' tasto n, vediamo che codice = n—1.

* In questo caso "chiaro" = meno saturo, cioè mischiato a del grigio.

Ad eccezione dell'arancione e dei grigi, se si ammette che rosa = rosso chiaro, si vede che chiaro = codice + 8.

Si nota che all'accensione tutto si svolge come se ci fosse POKE 53281,14: POKE 53280,6 (quadro azzurro, sfondo blu).
Dobbiamo notare che parecchi messaggi si possono ottenere almeno in due modi: ad esempio, per ottenere una A porpora su sfondo verde, potete mettere uno sfondo verde, poi stampare "Pur^c A". Ma potete mettere anche uno sfondo porpora e stampare "Grn^c Rvs^c A".

Esercizio 6.12 (macabro). *Preparare una partecipazione di morte.*

Esercizio 6.13 *Vorremmo la Francia in giallo su sfondo blu oceano.*

Ottenimento di colori mediante *POKE* nella memoria di schermo

Abbiamo già visto che il C64 possiede due memorie di schermo di 1000 bytes (1000 = 25 linee × 40 colonne). Ogni byte corrisponde ad una posizione sullo schermo. La prima memoria (che inizia in 1024) contiene i codici-carattere di ogni carattere da stampare. La seconda (che inizia in 55296) è, in realtà, composta da quartetti (sono presenti soltanto i quattro bits a destra). Ciascuna di queste celle contiene un

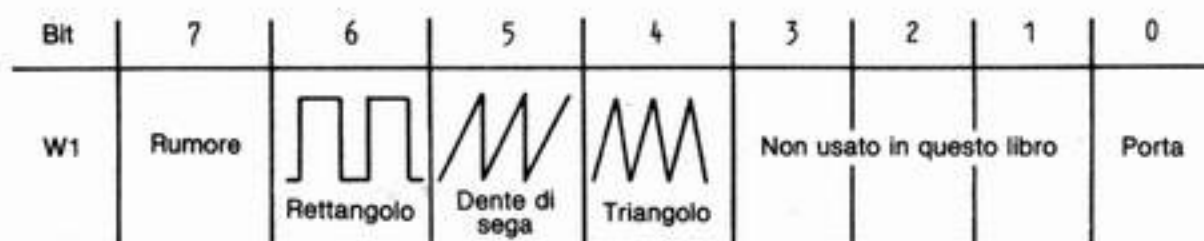
codice su quattro bit che determina il colore di stampa del carattere che si trova nella posizione corrispondente sullo schermo.
Qual è la corrispondenza tra il codice ed il valore? Ci sono due casi in funzione del bit 4 dell'indirizzo 53270.

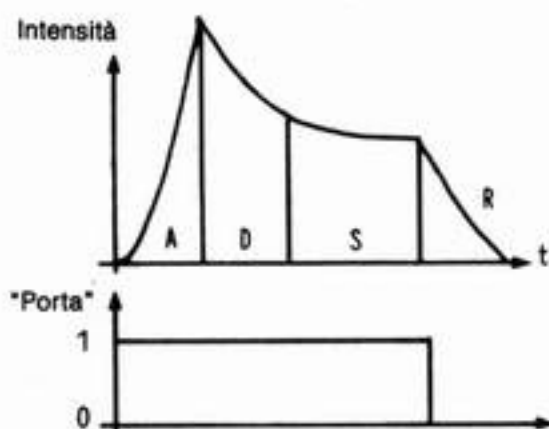
—Bit = 0. È il modo più normale. Il codice colore di stampa del carattere segue la corrispondenza già vista (tabella 6.4): O: nero, 1: bianco, 2: rosso... 7: giallo... 15: grigio chiaro).
In questo modo, possiamo quindi avere due colori in una casella schermo: o il colore dello sfondo (16 possibilità), oppure il colore carattere determinato dal codice (16 possibilità).
—Bit = 1. È un modo più complicato (modo multicolore) che non verrà studiato in questo libro, con l'aiuto del quale possiamo, in ogni maglia di schermo, avere 4 diversi colori.

Esercizio 6.14 *Disegnare la bandiera olandese sullo schermo. È più facile della bandiera italiana. Perché?*

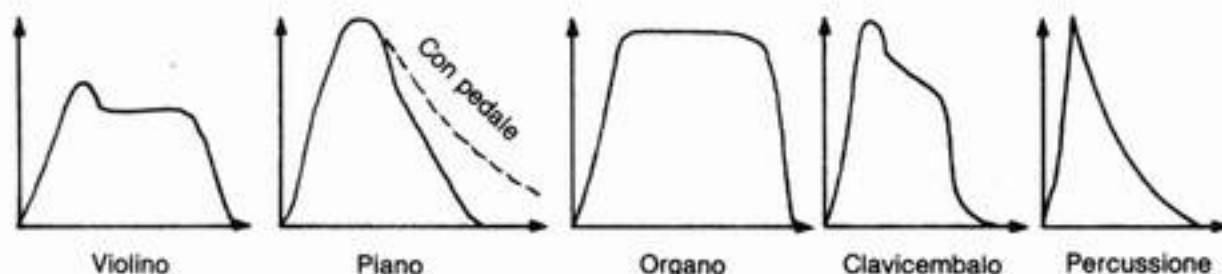
EFFETTI SONORI

Eccoci ad una sezione che farà molto chiasso! Infatti il C64 è capace di comandare l'altoparlante del televisore al quale è collegato per produrre suoni. Per tutta la sezione, regolate il volume del suono del vostro televisore ad un valore confortevole (suggeriamo 1/3 del massimo). Se il volume è a zero, non sentirete alcun suono e quindi, crederete che i vostri programmi sonori non funzionano o, altrimenti, che gli effetti sonori del vostro C64 siano guasti. Ciò può succedere però, prima di supporlo, verificate sul vostro televisore.
Per produrre suoni, il C64 è fornito di un circuito integrato particolare, il SID 6581 (Sound Interface Device: circuito d'interfaccia sonora) che è un autentico sintetizzatore. Il processore lo vede come un insieme di registri. È estremamente complesso e, in questo libro, vedremo soltanto gli elementi più semplici.
Si ottengono gli effetti sonori mettendo dei valori nei registri, con POKE, agli indirizzi voluti. I registri si dividono in registri generali e registri associati alle tre "voci" capaci di funzionare simultaneamente. Le tre voci funzionano pressapoco allo stesso modo.
Per ogni voce, si possono controllare i seguenti parametri (citiamo qui di seguito i nomi simbolici per la voce 1):

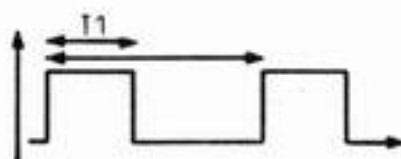
— la frequenza, sotto forma di un numero di 16 bits, dunque due registri L1 (parte bassa) e H1 (parte alta);
— la forma d'onda triangolare, dente di sega, rattangolo e rumore bianco. Il registro che corrisponde alla voce 1 ha per nome simbolico W1. Obbedisce allo schema seguente:

Per attivare un suono di una data forma, bisogna assegnare 1 al bit corrispondente e assegnare 1 a "porta". Con "porta" a 0, si entra nella fase di rilassamento del suono.

— l'involucro del suono. L'intensità di un suono passa da 4 fasi (figura qui di fronte) che determinano l'involucro del suono o ADSR:

A = attacco (crescita del suono);
D = discesa verso la soglia;
S = sostegno a soglia;
R = rilassamento;

A e D occupano ciascuno 4 bits del registro Al.
A (bits 4 a 7) varia da 0 a 15, il che corrisponde ad una durata da 2 ms a 8 s;
D (bits 0 a 3) esprime una durata da 5 ms a 24 s;

S e R occupano ciascuno 4 bits del registro S1. S (bits 4 a 7) da 0 a 15 esprime il livello di soglia da 0 a 100% rispetto alla vetta della salita. Ad esempio, nella figura qui sotto, la soglia è a mezza altezza della vetta: avremo dunque S = 8. R occupa i bits da 0 a 3 ed esprime una durata di decrescita da 6 ms a 25 s. R è poco usato in pratica: non ci si occupa di fermare un suono: si fa partire la nota successiva.
L'inviluppo è molto importante per l'imitazione degli strumenti musicali: la figura qui sotto dà qualche esempio

Quando la forma d'onda è rettangolare, bisogna definirne il rapporto ciclico (TI/T nella figura qui di seguito). Tale rapporto è definito da

un numero di 12 bits espresso in 1/4096 esimi, riposto nei registri Cl (parte bassa) e Dl (parte alta).

Dunque ci sono 7 registri per voce, ossia 21 registri ai quali vengono ad aggiungersi 8 registri generali (che agiscono su tutte le voci).
In questo libro, useremo soltanto il registro V i cui bits 0 e 3 definiscono il volume del suono (0 = silenzio, 15 = suono più forte possibile).
Gli indirizzi sono indicati nella tabella seguente:

Tab. 6.5. **Registri sonori del C64.**

Nome Simbolico	Indirizzo	Funzione	
L1	54272	Frequenza (parte bassa)	voce 1
H1	54273	Frequenza (parte alta)	
C1	54274	Rapporto ciclico (p. bassa)	
D1	54275	Rapporto ciclico (p. alta)	
W1	54276	Forma d'onda	
A1	54277	Attacco/discesa	
S1	54278	Sostegno rilassamento	
L2	54279	Frequenza (parte bassa)	voce 2
H2	54280	Frequenza (parte alta)	
C2	54281	Rapporto ciclico (p. bassa)	
D2	54282	Rapporto ciclico (p. alta)	
W2	54283	Forma d'onda	
A2	54284	Attacco/discesa	
S2	54285	Sostegno rilassamento	
L3	54286	Frequenza (parte bassa)	voce 3
H3	54287	Frequenza (parte alta)	
C3	54288	Rapporto ciclico (p. bassa)	
D3	54289	Rapporto ciclico (p. alta)	
W3	54290	Forma d'onda	
A3	54291	Attacco/discesa	
S3	54292	Sostegno rilassamento	
FB	54293	Frequenza filtro (3 bit a destra)*	
FH	54294	Frequenza filtro (bits 3-10)*	
RF	54295	Risonanza/filtraggio*	
V	54296	Bits 0-3: volume; bits 4-7: modo filtraggio*	
PX	54297	Lettura potenziometro X*	
PX	54298	Lettura potenziometro Y*	
O3	54299	Lettura oscillatore voce 3*	
E3	54300	Lettura involucro voce 3*	

* Non usato in questo libro.

Per ottenere un suono, la sequenza di operazioni è ora abbastanza semplice:

1. definire gli indirizzi simbolici usati: L = ... H = ... V = ... ;
2. definire la frequenza con POKE L, ... POKE H, ... seguendo la tabella 6.2. Definire il rapporto ciclico se necessario;
3. definire l'involucro con POKE A, ... POKE S, ...;
4. fissare un valore del volume, ad esempio POKE V, 10;
5. lanciare il suono con POKE W, ... con "porta" = 1, quindi 17: triangolo, 33: dente di sega, 65: rettangolo, 129: rumore;
6. generare l'intervallo di durata della nota (con un ciclo o con TI o qualsiasi altro metodo);
7. iniziare la fase di rilassamento con POKE W, ... con "porta" = 0, quindi valore 16, 32, 64 o 128;
8. fermare tutto con POKE 0 a tutti gli indirizzi.

Note: *l'operazione 5 lancia il suono. Questo continua poi conformemente al proprio inviluppo fino all'arresto con POKE 0 (o STOP RESTORE) o inizio di un altro suono. Questo permette di lanciare simultaneamente fino a tre suoni per fare degli accordi.*
In pratica, quando si fanno sequenze di note, non si effettua l'operazione 7: si inizia il suono successivo. Prima dell'inizio di qualsiasi suono, è preferibile azzerare il SID con un'operazione 8.
Si possono mischiare due forme d'onda in W (esempio 97), ma il rumore non deve essere mischiato.

La tabella 6.6 dà i valori delle note (valori approssimativi poiché dipendono dalla frequenza dell'orologio e dallo standard televisivo).

Esempio: ecco un programma che fa trattenere il suono fino alla pressione di un tasto qualsiasi. Usiamo la voce 1.

```
10 V=54296:W=54276:A=54277:H=54273:L=54272:S=A+1
20 POKEV,5  :POKEW,33:POKEA,31:POKES,248
30 POKEH,17:POKEL,75
40 GETA$:IF A$="" GOTO40
50 POKEW,0:POKEA,0:POKES,0:POKEV,0
```

Valori da sperimentare

W	A		W	A	
33	31	Tromba	17	102	Gong
97	31	Oboe (miscuglio di 2 forme d'onda)	17	0	Organo
17	31	Flauto	33	9	Clavicembalo (S=O)
33	239	Violino	65	9	Piano (S=O)

Tab. 6.6. **Note sul C64.**

Nota	L	H	Nota	L	H	Nota	L	H
Do –0	18	1	Do –2	73	4	Do –4	37	17
Do #	35	1	Do#	139	4	Do#	42	18
Re	52	1	Re	208	4	Re	63	19
Re#	70	1	Re#	25	5	Re#	100	20
Mi	90	1	Mi	103	5	Mi	154	21
Fa	110	1	Fa	185	5	Fa	227	22
Fa#	132	1	Fa#	16	6	Fa#	63	24
Sol	155	1	Sol	108	6	Sol	177	25
Sol#	179	1	Sol#	206	6	Sol#	56	27
La	205	1	La	53	7	La	214	28
La#	233	1	La#	163	7	La#	141	30
Si	6	2	Si	23	8	Si	94	32
Do –1	37	2	Do –3	147	8	Do –5	75	34
Do#	69	2	Do#	21	9	Do#	85	36
Re	104	2	Re	159	9	Re	126	38
Re#	140	2	Re#	60	10	Re#	200	40
Mi	179	2	Mi	205	10	Mi	52	43
Fa	220	2	Fa	114	11	Fa	198	45
Fa#	8	3	Fa#	32	12	Fa#	127	48
Sol	54	3	Sol	216	12	Sol	97	51
Sol#	103	3	Sol#	156	13	Sol#	111	54
La	155	3	La	107	14	La	172	57
La#	210	3	La#	70	15	La#	126	61
Si	12	4	Si	47	16	Si	188	64
Do –6	149	68	Sol#	223	108	Mi	210	172
Do#	169	72	La	88	115	Fa#	25	183
Re	252	76	La#	52	122	Fa	252	193
Re#	161	81	Si	120	129	Sol	133	205
Mi	105	86	Do –7	43	137	Sol#	189	217
Fa	140	91	Do#	83	145	La	176	230
Fa#	254	96	Re	247	153	La# –7	103	244
Sol	194	102	Re#	31	163			

Per il violino, bisognerebbe aggiungere del vibrato.
Per il piano, fissare il rapporto ciclico a 1/2 con un POKE L+3,4.

Esercizio 6.15 *Nel gioco del quiz geografico, quando il giocatore ha vinto, fate squillare le prime note della marsigliese. (re re, re sol, sol la, la re, si sol).*

Esercizio 6.16 *Fate sentire il cucù nel programma C.4.*

Esercizio 6.17 *(gioco di SIMON — marchio depositato). Si stabilisce una corrispondenza tra i tasti colore e le note della gamma. Il pro-*

gramma sceglie a caso una sequenza di 4 note: la fa suonare, mostrando contemporaneamente, su tutto lo schermo, i colori corrispondenti. Il giocatore deve quindi battere la sequenza corrispondente. Mentre preme un tasto, il programma suona la nota e visualizza il colore. Si autorizzano due tentativi per trovare una sequenza.

Il generatore di rumori

Con W = 129, non si generano note ma rumori, cioè miscugli casuali di frequenze (intorno ad una frequenza determinata dai numeri posti in L e H).
Per giudicare gli effetti che possono essere ottenuti, la cosa migliore è sperimentare con il programmino che segue: esso genera un rumore e stampa il numero corrispondente a L + 256 * H. Tale numero cresce se premete "+" e decresce se premete "—":

PROVA DI RUMORI

```
10 PRINT"♫":POKE650,128
15 V=54296:L1=54272:H1=L1+1:W1=L1+4:A1=W1+1:S1=A1+1
20 X=32000:INC=100:POKEV,10:POKEA1,9:POKES1,240
25 XH=INT(X/256):XL=X-256*XH
27 PRINT"■                                    "
30 PRINT"■";X;XL;XH;"                         "
40 POKE L1,XL:POKEH1,XH:POKE W1,129
50 GETA$
60 IFA$="-"THENIF X>INC THEN X=X-INC:GOTO25
70 IFA$="+"THENIF X<65535-INC THEN X=X+INC:GOTO25
80 IFA$=" |"THEN INC=INC-10:GOTO25
90 IFA$="+"THEN INC=INC+10:GOTO25
100 GOTO25
```

Il POKE 650 assicura che tutti i tasti, dunque "+" e "—" avranno ripetizione automatica. Si vede quindi che:

intorno a	abbiamo
200-300	rombi, macchine, esplosioni, aerei
1000	vento, onde
1300	tempesta, lancia-fiamme
1500	blizzard
2500	reattori, fughe di gas (3500)
10000	fischi

Non si dimentichino gli effetti dinamici per variazione delle frequenze o l'effetto di decrescita (sparo se A = 23, S = 0).

Esercizio 6.18 *Un aereo si avvicina, ci sorvola, poi si allontana.*

Abbiamo, ormai, esplorato più o meno tutte le risorse spettacolari del C64. Ci rimane il tracciato di figure, l'animazione e l'alta risoluzione.

Capitolo 7

Figure e disegni animati Grafici alta risoluzione

Prima di affrontare i disegni animati veri e propri, realizzeremo un programma "serio", importantissimo per le sue applicazioni: il tracciato del grafico di una funzione.

GRAFICO DI UNA FUNZIONE - ISTRUZIONE *DEF FN*

Il programma non è, in realtà, complicatissimo, Tracceremo il grafico della funzione esponenziale $y = \exp(x)$, (e^x) per x che va da 0 a 1. Il grafico verrà tracciato punto per punto. Disponendo di 25 linee sullo schermo, potremo rappresentare 23 punti.
Sulla linea i verrà materializzato il punto corrispondente ad $x = i/24$.
L'asse delle ascisse in questo caso è verticale, orientato da sinistra verso destra. Esso può variare da 1 a 40. Su di una linea data, verrà stampata una stella nella colonna il cui numero è proporzionale al valore di y per l'x corrispondente alla linea.
Bisogna dunque applicare alla funzione un fattore di scala, affinché vari tra 1 e 40 di modo da coprire tutto lo schermo.
Per la funzione exp, useremo:

$$y = 39.(\exp(x)-1)/(e-1)+1$$

La formula generale per una funzione f(x) il cui massimo e minimo nell'intervallo di variazione studiato sono rispettivamente max e min sarebbe:

$$y = 1 + (39.(f(x)-min))/(max-min)$$

Per stampare il grafico, basta quindi realizzare un ciclo FOR X = inizio TO fine STEP intervallo, e per ogni punto una stella nella colonna INT(y).

Basic dispone di due funzioni semplici a questo scopo:

PRINT TAB(N); posiziona sulla colonna N
PRINT SPC(N); fa stampare N spazi dunque posiziona su N+1

Il programma da realizzare ne deriva direttamente:

PROGRAMMA C.7A

```
20 FOR X=0 TO 1 STEP 1/24
30 N=1+INT(39*(EXP(X)-1)/(EXP(1)-1))
40 PRINT TAB(N);"*"
50 NEXT
60 GOTO 60
```

Naturalmente, un programma veramente efficiente dovrebbe offrire una presentazione migliore: titolo, disegno degli assi, coordinate, ecc. Avete tutte le conoscenze necessarie per farlo, e vi suggeriamo di esercitarvici.

Esercizio 7.1 *Tracciate due curve sullo stesso grafico, ad esempio sin(x) e cos(x) per x da 0 a 2π.*

Istruzione *DEF FN*

Può accadere di dover tracciare il grafico di una funzione che non è esattamente una delle funzioni della biblioteca, ma una combinazione di diverse di esse.
Questa combinazione può intervenire più volte nel programma. In tal caso è noioso riscriverla ogni volta.
Esiste una istruzione che permette di risolvere questo problema: si tratta di una istruzione che permette all'utente di definire le proprie funzioni, che vanno allora ad aggiungersi alle funzioni della biblioteca.

Esempi:

```
10 DEF FNHARM (X)=(A*X+B)/(C*X+D)
20 DEF FNA(X)=L*sin(X)+cos(2*X)
30 DEF FNF(T)=EXP(—T↑2/2)
40 DEF FNG(X)=1+20*(FN F(X)—MIN)/(MAX—MIN)
```

Il nome della funzione definita segue FN; obbedisce alle abituali regole applicabili ai nomi di variabili.
Un richiamo della funzione FNHARM definita in 10 potrebbe essere 100 Z = 1 + FNHARM(3.5).

L'espressione data nella definizione viene quindi calcolata, sostituendo l'argomento formale con il valore (qui 3.5) fornito al momento della chiamata. Per le altre variabili che intervengono (qui A, B, C e D), è dei loro valori al momento della chiamata che viene tenuto conto. L'argomento può essere lui stesso fornito sotto forma di una qualsiasi espressione da calcolare:

100 PRINT FNA(U * A + EXP(B)/C)

Gli esempi 30 e 40 qui sopra mostrano che è possibile usare, nella definizione di una funzione, una funzione già definita.
Usando questa nuova possibilità, ecco il programma C.7B che traccia il grafico della funzione di Gauss tra A e B:

PROGRAMMA C.7B

```
5 INPUT"ESTREMI";A,B
10 DEF FNF(X)=EXP(-X↑2/2)
20 DEF FNG(X)=1+38*(FNF(X)-MIN)/(MAX-MIN)
30 MAX=-10↑38:MIN=10↑38
40 FOR X=A TO B STEP (B-A)/22
50 IF FNF(X)>MAX THEN MAX=FNF(X)
60 IF FNF(X)<MI THEN MIN=FNF(X)
70 NEXT X
80 FOR X=A TO B STEP (B-A)/22
90 N=INT(FNG(X))
100 PRINT TAB(N);"*":NEXT X
110 GOTO 110
```

Il vantaggio del programma C.7B rispetto alla versione precedente è che è generale: gli estremi vengono forniti da INPUT, e per cambiare funzione da tracciare, basta ribattere l'istruzione 10.
Da 30 a 70 vengono effettuati i calcoli del massimo e del minimo della funzione, allo stesso modo che nell'esercizio 5.10.
In 60 abbiamo scritto MI anziché MIN; sappiamo che ciò rimanda alla stessa variabile. Se avessimo scritto MIN, Basic avrebbe letto IF FNF(x) < M <u>INT</u> HEN... ed avrebbe trovato la parola chiave INT in posizione errata. Ecco il tipo di piccole sorprese che si possono avere!
Abbiamo incontrato nei programmi della serie B che precedeva, una buona occasione di usare un DEF FN; cioè per stampare un risultato conservando soltanto due cifre decimali: la stessa espressione torna varie volte.
Ad esempio, in B.8, avremmo potuto scrivere:

3 DEF FNZ(X) = INT(X * 100)/100

Le linee 60 e 90 diventano rispettivamente:

```
60 PRINT "ERRORE"; FNZ(E); "%": GOTO 30
90 PRINT "VINTO IN"; FNZ(SC(J)); "SECONDI"
```

Esercizio 7.2 *Riprendete il programma dell'esercizio 5.4. Normalmente, l'avete salvato su cassetta. Avete una popolazione spartita fra 10 classi. Tracciate l'istogramma corrispondente.*

Un istogramma è un diagramma formato da tante sbarre quante classi, con le sbarre di lunghezza proporzionale al numero di elementi della classe corrispondente. Ad ogni linea, non sono N spazi che bisogna stampare, ma N Rvs^c Sp.

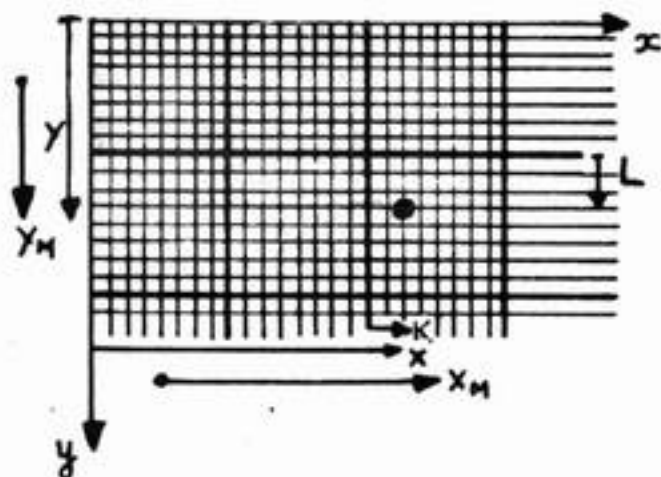
Il problema della risoluzione

Tutti i grafici che abbiamo ottenuto sono approssimativi. Infatti, si può procedere soltanto per numeri interi di caselle di stampa: la risoluzione è 25 × 40. È stato possibile migliorare la risoluzione apparente nella carta di Francia, scegliendo con cura tra i caratteri grafici usati. Per contro, per il posizionamento delle città, siamo rimasti al limite dei 25 × 40.
Per il tracciato di grafici, esiste un modo per approfittare di una migliore risoluzione guadagnando un fattore di 2 in ogni direzione. Per questo, si devono utilizzare i caratteri grafici ▘ ▝ ▖ ▗ e ▚ . Con i loro inversi, essi permettono di ottenere un'intreccio di 50 × 80 ossia 4000 punti per schermo. Vi suggeriamo di provare a tracciare grafici, per mezzo di questi caratteri, ma si tratta di un esercizio delicato!
Ma il C64 possiede, in realtà, il modo di fare dei grafici con la massima risoluzione permessa dallo schermo di televisione: si guadagna un fattore di 8 in ogni direzione poiché la risoluzione diviene 200 × 320.

GRAFICA ALTA RISOLUZIONE

Per fare della grafica alta risoluzione, dobbiamo capire come funziona il metodo di visualizzazione del C64. Si tratta di una questione delicata che cercheremo di semplificare al massimo. Possiamo considerare che lo schermo del C64 (la parte attiva, cioè che sta all'interno del quadro) è formato da punti elementari, ognuno dei quali è suscettibile di essere acceso o spento. Qui, "spento" vuol dire "del colore che è stato scelto per lo sfondo dello schermo (da POKE 53281)", "acceso" vuol dire "di un colore determinato dal contenuto della zona di memoria 55296...". Questi punti formano una rete di 200 linee elementari su 320 colonne elementari. Il televisore spazza l'immagine, linea elementare per linea elementare. Per ogni punto dell'immagine, al

momento in cui la spazza, esso deve essere capace di sapere se è acceso o spento. Il fatto di essere acceso o spento potendo codificarsi in un bit (1 = acceso, 0 = spento), ciò rappresenta un'informazione di $200 \times 320 = 64000$ bits = 8000 bytes o 8K. Il tempo necessario per riempire questi 8K bytes, per formare un'immagine, è lungo. Quindi, per i messaggi correnti, bisogna semplificare.
A questo scopo, si raggruppano i punti elementari in rettangoli di 8 linee elementari per 8 colonne elementari chiamati "maglie". Un carattere viene a stamparsi sullo schermo in una maglia. Le maglie sono suddivise in linee e colonne. Bisogna distinguere bene queste linee e colonne che chiameremo "macroscopiche" dalle linee e colonne elementari. Si vede che lo schermo è formato da 25 linee per 40 colonne (macroscopiche).
Dato un punto di coordinate elementari x e y (assi come nella figura qui di fronte origine di numerazione 0), le sue macrocoordinate sono:

$$XM = INT(X/8) \quad e \quad YM = INT(Y/8)$$

Così il punto messo in rilievo nella figura ha per coordinate elementari X = 18; Y = 11. Appartiene alla maglia di coodinate XM = 2; YM = 1. Rispetto alla maglia, sta sulla linea elementare L = 3 (resto della divisione di Y per 8), colonna elementare K = 2 (resto della divisione di X per 8).
Ora notiamo che i caratteri che si possono stampare in una maglia appartengono ad un gruppo ristretto: mentre ci sono 2^{64} combinazioni possibili (ci sono 64 punti in ogni maglia, che possono essere accesi o spenti), in realtà si visualizzano soltanto 256 caratteri possibili (ivi compresi i grafici e i contrasti invertiti). Codificheremo quindi l'informazione in due tappe.
In una prima memoria (obbligatoriamente viva), dal numero di bytes uguale al numero di maglie sullo schermo (dunque 1000 per il C64), metteremo il codice del carattere che deve figurare in ogni maglia.

La E è l'indirizzo di origine di questa memoria (1024), l'indirizzo corrispondente alla maglia di coordinate XM, YM è: Q = E + 40 * YM + XM (la formula è leggermente diversa da quella di pagina 71 perché, in questo caso, numeriamo da 0 in poi).
Nell'esempio qui sopra, avremmo: Q = 1024 + 24 = 1048. Se in quella maglia, si vuole visualizzare A, vi si metterà il codice di A (cioè 1, secondo la tab. 6.2).
Abbiamo poi una seconda memoria (che può essere memoria morta) che conterrà la corrispondenza codice-disegno. Questa memoria si chiama il generatore di caratteri, e la sua ampiezza è di 256 × 8 = 2K bytes.
Chiamiamo G la sua origine (53248 sul C64 all'accensione). Ci sono, in questa memoria, 8 bytes per ogni carattere. All'indirizzo G + 8 * R + L, si trova un byte che è l'immagine della linea elementare L che ci vuole per disegnare il carattere di codice schermo R.
Provate il programma:

```
5 POKE56334,PEEK(56334)AND254
6 POKE1,PEEK(1)AND251
10 G=53248:R=1:REM CODICE DI 'A'
20 FOR L=0 TO 7
30 A(L)=PEEK(G+8*R+L):NEXT
35 POKE1,PEEK(1)OR4
36 POKE56334,PEEK(56334)OR1
40 FOR L=0 TO 7
50 PRINTA(L):NEXT
```

Ottenete le stampe:

```
 24  =  00011000   ossia   . . . ● ● . . .
 60  =  00111100           . . ● ● ● ● . .
102  =  01100110           . ● ● . . ● ● .
126  =  01111110           . ● ● ● ● ● ● .
102  =  01100110           . ● ● . . ● ● .
102  =  01100110           . ● ● . . ● ● .
102  =  01100110           . ● ● . . ● ● .
  0  =  00000000           . . . . . . . .
```

Si vede come ciò fa disegnare una A.
Il C64 offre due modi di fare alta risoluzione.

Si può, con un POKE, in un registro della cassetta VIC, dire che il generatore di caratteri è situato in RAM. Si possono quindi definire le forme che si vogliono. In pratica, applicheremo ciò alla creazione di altri gruppi di caratteri.
Questo permette, ad esempio, le lettere greche, certi segni matemati-

ci, i caratteri grassetti o italici per un mini trattamento di testi, oppure permette di aggiungere altri caratteri grafici a quelli che vengono forniti per creare insetti, invasori o altri...
— Si può sempre, con POKE, far visualizzare in modo alta risoluzione nel quale si definiscono, bit per bit, i punti accesi o spenti. Ovviamente, facendolo, si rinuncia alla semplificazione citata prima e diventano necessari 8000 bytes di memoria (cosa che non dà fastidio in un C64).

Modifica del generatore di caratteri

Come esempio di modifica del generatore di caratteri, creeremo la lettera greca *α* invece del carattere @. Lo stesso metodo varrebbe per ogni altro carattere e servirebbe, ad esempio, per creare la à, la è, ecc.
Si devono fare quattro cose:
— scegliere la zona di RAM dove andrà il generatore e proteggere tale zona da Basic;
— copiare il generatore di carattere ROM nella zona RAM, tranne se si cambiano tutti i caratteri;
— creare, in RAM, i nuovi caratteri desiderati;
— dire al C64 che, ora, il generatore di caratteri sta all'indirizzo RAM che era stato scelto.
Per tornare alla situazione di partenza, quando abbiamo finito di usare i nuovi caratteri, basta fare STOP/RESTORE.
Vediamo, ora, le tappe una per una:
— L'indirizzo di inizio del generatore di caratteri RAM, il più facile da adottare, è 12288 (3000 in esadecimale).
Per proteggere tale zona, bisogna dire al Basic che la sua memoria si ferma lì, con POKE 52,48: POKE 56,48: CLR.
Ovviamente, ciò riduce parecchio la memoria lasciata a Basic (11K invece di 39), ma:
● basta per parecchie applicazioni e, in ogni caso, per quelle di questo libro;
● esistono altre possibilità che verranno spiegate nel prossimo libro di questa serie.
— Si usa una tecnica analoga a quella della pagina precedente:

```
5 POKE 56334, PEEK (56334) AND 254
6 POKE 1, PEEK(1) AND 251
10 Gl = 53248:G2 = 12288
20 FOR I = 0 TO 2047
30 POKE G2 + I, PEEK(G1 + I):NEXT
35 POKE 1, PEEK(1) OR 4
36 POKE 56334, PEEK(56334) OR 1
```

Si può anche copiare soltanto una parte del generatore di caratteri

cambiando i limiti del ciclo in 20 (infatti, la copiatura completa è lunga).
— Si procede in binario in una griglia 8 × 8, come abbiamo visto per A. Se R è il codice del carattere da modificare (qui 0):

	Esa	Dec	Indirizzo G2 + 8*R +
. . . .	0	0	0
. . . .	0	0	1
. . ● ● ● . ● ●	3B	59	2
. ● ● ● ● ● ● .	7E	126	3
. ● ● . ● ● . .	6C	108	4
. ● ● . ● ● . .	6C	108	5
. ● ● ● ● ● ● .	7E	126	6
. . ● ● ● . ● ●	3B	59	7

e quindi:

```
40 DATA 0, 0, 59, 126, 108, 108, 126, 59
45 R=0
50 FOR L=0 TO 7:READ A:POKE G2+8*R+L, A:NEXT
```

— Si attiva il generatore di caratteri in 3000 esa con

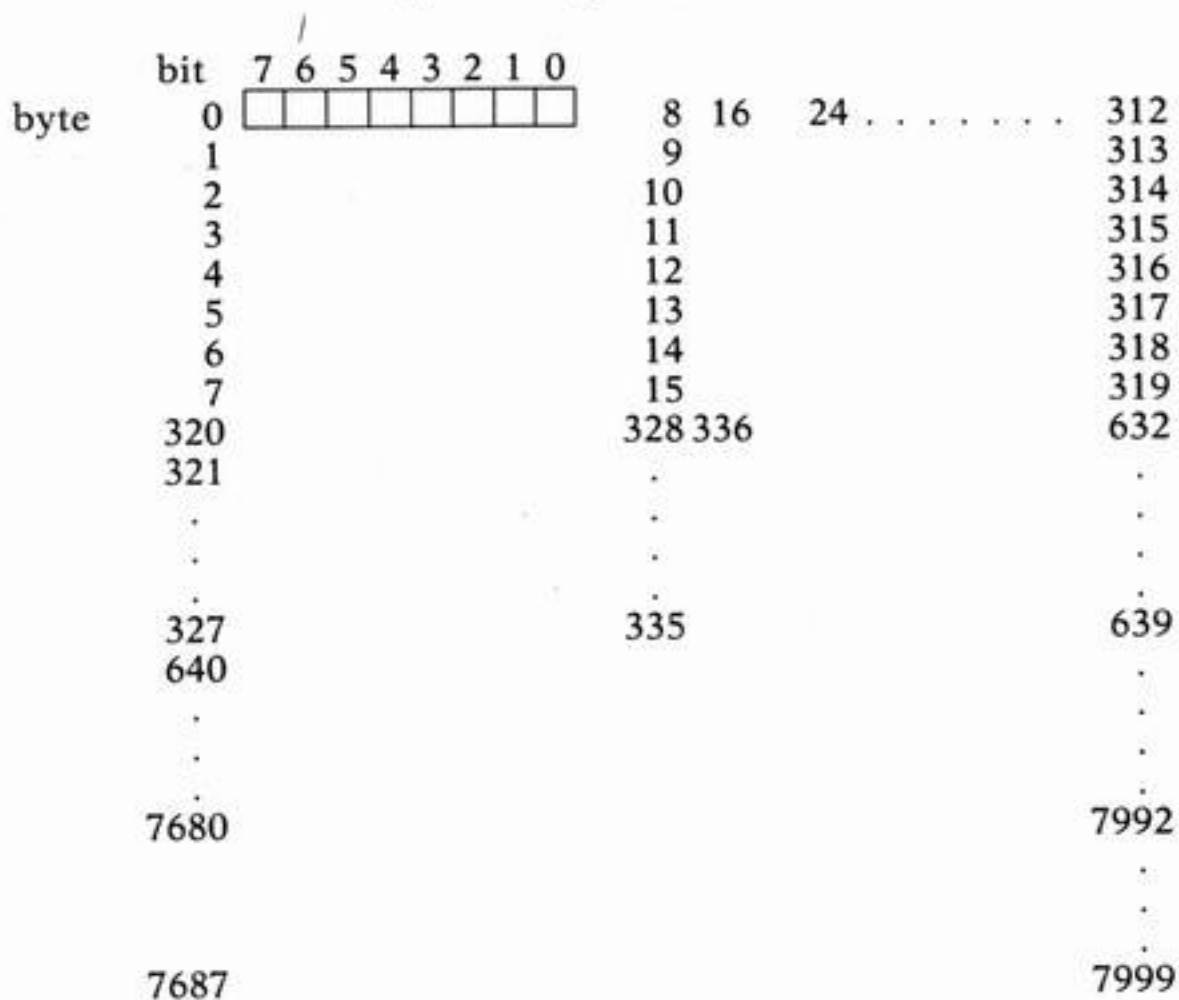
```
POKE 53272, (PEEK(53272)AND240)+12
```

(impone 1100 nei bits 0 a 3 del registro d'indirizzo 53272)

Esercizio 7.3 *Riunite quello che precede e fate stampare : α = ALFA.*

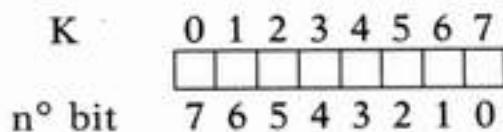
Tracciato di grafici in alta risoluzione

Quando il C64 viene messo in modo alta risoluzione, utilizza la memoria in maniera diversa dal modo carattere. La memoria di colore (55296...) non viene utilizzata. Il colore verrà codificato in 1024 e seguenti (la vecchia memoria di schermo).
Ogni byte specifica due colori; bits 0-3: colore dello sfondo (punti "spenti"); bits 4-7: colore del grafico (punti "accesi").
Ciò permette di specificare i colori in ogni "maglia" dello schermo ma, in pratica, specificheremo gli stessi dappertutto.
Ad esempio, FOR I = 1024 TO 2023: POKE I, 1: NEXT specificherà un grafico nero su sfondo bianco.
Gli 8000 bytes che specificano i punti accesi o spenti devono essere presi in un'altra zona memoria di origine E. I bytes di quella memoria sono numerati da E + 0 ad E + 8000, secondo lo schema qui di seguito che li suddivide in maglie caratteri.

```
             bit   7 6 5 4 3 2 1 0
byte           0  [ | | | | | | | ]    8  16   24 . . . . . . .  312
               1                       9                          313
               2                      10                          314
               3                      11                          315
               4                      12                          316
               5                      13                          317
               6                      14                          318
               7                      15                          319
             320                     328 336                      632
             321                       .                            .
               .                       .                            .
               .                       .                            .
               .                       .                            .
             327                     335                          639
             640                                                    .
               .                                                    .
               .                                                    .
               .                                                    .
            7680                                                 7992
                                                                    .
                                                                    .
                                                                    .
            7687                                                 7999
```

Secondo questo schema date le coordinate di un punto elementare X, Y, si determina in quale maglia elementare XM, YM esso sta con:

$XM = INT(X/8)$ e $YM = INT(Y/8)$
$0 \leq X \leq 319 \quad 0 \leq Y \leq 199 \quad 0 \leq XM \leq 40 \quad 0 \leq YM \leq 25$

Il punto sta, nella propria maglia, sulla linea elementare L = YAND7 (altra formula più lunga da calcolare $L = Y - 8 * YM$) ne segue l'indirizzo del byte in questione: $Q = E + 8 * (40 * YM + XM) + L$. In questo byte, il punto sta nella colonna elementare: K = X AND 7(o $X - 8 * XM$).
Il numero del bit corrispondente è 7—K, come mostra lo schema:

```
K        0 1 2 3 4 5 6 7
        [ | | | | | | | ]
n° bit   7 6 5 4 3 2 1 0
```

Quindi, per accendere un punto, bisogna fare:

```
POKE Q, PEEK(Q) OR 2↑(7-K)
```

e per spegnere lo stesso punto, bisogna fare:

```
POKE Q, PEEK(Q) AND (255-2↑(7-K))
```

Cosa ci manca per un tracciato effettivo?

— Scegliere l'origine della memoria di stampa e specificarla al C64. Per questo libro, la scelta migliore è E = 8192 (esadecimale 2000). La si specifica con:

POKE 53272, PEEK(53272) OR 8

Fatto questo, dobbiamo dire al C64 che non crei variabili Basic sopra 8192. Lo si fa con le istruzioni POKE 56, 32: POKE 52, 32: CLR. In 55, 56 e 51, 52, si trovano dei puntatori che contengono l'indirizzo più alto accessibile da Basic. Li trasformiamo affinché Basic "creda" che la sua memoria si ferma in 2000 esa (32 dec = 20 esa). Il CLR azzera tutte le variabili: lo si deve fare quando si sono appena trasformati i puntatori.
Ciò implica le stesse note che a pagina 101. La zona di memoria, lasciata a Basic, è molto ristretta. Altri indirizzi, che lasciano più posto, sono possibili: verranno descritti nel prossimo libro di questa serie.
— Riempire la memoria di colore HR, ad esempio con degli 1 (figura nera su sfondo bianco).
Riempire la memoria di stampa con degli 0: fintantoché non abbiamo tracciato nulla, ci vuole lo schermo vuoto e non lo si può fare con ?"clr" che agisce soltanto in modo caratteri.
— Mettere il C64 in modo HR con POKE 53265, PEEK(53265) OR 32 (si ritorna in modo normale con POKE 53265, PEEK(53265) AND 223).
Per accendere il punto di coordinate X, Y, basta quindi richiamare i sottoprogrammi 900 che calcola le coordinate di maglia:

900 XM = INT(X/8): YM = INT(Y/8): P = E + 320 * YM + 8 * YM: RETURN

e 1000 che accende il punto:

1000 K = X AND 7: L = Y AND 7: Q = P + L
1010 POKE Q, PEEK(Q) OR 2↑(7-K): RETURN

Siamo pronti, ormai, a tracciare una sinusoide. X varia da 0 a 319, Y da 0 a 199, quindi prenderemo la formula: Y = 100 + 90 * SIN(8 * π * X/318) affinché la figura abbia 4 alternanze.

PROGRAMMA D.1

```
10 POKE53272,PEEK(53272) OR 8
20 POKE52,32:POKE56,32:CLR
30 E=8192:CO=1024
40 FOR I=CO TO CO+999:POKE I,1:NEXT
50 FOR I=E TO E+7999:POKEI,0:NEXT
60 POKE53265,PEEK(53265) OR 32
```

```
70 DEF FNF(X)=100+90*SIN(8*π*X/318)
80 FOR X=0 TO 318
90 Y=FNF(X)
100 GOSUB900:GOSUB1000:NEXT
110 GOTO110
900 XM=INT(X/8):YM=INT(Y/8):P=E+320*YM+8*XM:RETURN
1000 K=X AND7:L=Y AND 7:Q=P+L
1010 POKE Q,PEEK(Q) OR 2↑(7-K):RETURN
```

Esercizio 7.4 *Fare un sottoprogramma 2000 che, anziché accendere il punto di coordinate X, Y, spenga quel punto.*

Esercizio 7.5 *Siamo ormai in grado di trasformare il nostro C64 in teleschermo. In funzione del tasto premuto spostiamo il punto di tracciato. Se il tasto CTRL non è premuto, avviene un semplice spostamento. Se il tasto CTRL è premuto simultaneamente, si scrive. L'indirizzo memoria 2 o 3 contiene in ogni momento l'immagine del tasto attualmente premuto. Vi proponiamo i seguenti assegnamenti:*

Tasto	PEEK(203)	Movimento	Tasto	PEEK(203)	Movimento
CRSR ⇓	7	↓	F1	4	↖
CRSR ⇒	2	→	F3	5	↗
CLR/HOME	51	↑	F5	6	↙
INST/DEL	0	←	F7	3	↘

Se nessun tasto è premuto PEEK(203) = 64
L'indirizzo memoria 653 contiene in ogni momento lo stato dei tasti SHIFT, C= e CTRL secondo la tabella:

Tasto	PEEK(653)
NULLA	0
SHIFT	1
C=	2
CTRL	4

Vediamo che nessun movimento necessita un SHIFT. Vogliamo conservare il fatto che SHIFT CLEAR/HOME svuoti lo schermo. È C= HOME che riporta lo spot all'origine, poiché HOME da solo fa ←.

Esercizio 7.6 *Torniamo al tracciato di figure (programma D-1). Non si potrebbero avere insieme la figura e dei caratteri (per dei titoli o le coordinate)?*

Ecco! Non ci rimane da dare che un po' di movimento.

Nozioni sui disegni animati

Ci limiteremo qui alle prime nozioni che permettono di animare ciò che il C64 visualizza.
Il primo metodo che viene in mente è il metodo classico utilizzato al cinema. La scena viene scomposta in un grande numero di disegni che vengono proiettati in successione.
Con il C64, che pure è uno dei microcomputer più veloci, si pone il problema del tempo di riempimento dello schermo per ottenere una immagine. Tanto più che, per non avere troppi scintillii, bisogna visualizzare almeno 15 immagini/secondo. Questo necessita il ricorso al linguaggio macchina che è studiato in un'altra opera di questa collana.
Per poter usare il Basic, bisogna passare da un'immagine alla successiva cambiando soltanto una parte molto piccola dell'immagine. Si possono usare, a scelta, dei PRINT o, meglio (più rapido), dei POKE nella memoria. È proprio quello che è stato realizzato nel programma del cucù (programma C.4, esercizio 6.9) che costituisce un embrione di disegno animato.

Movimenti di una palla

Come esempio, senza importanti modifiche da un'immagine alla seguente, faremo spostare una pallina sullo schermo. Ciò porterà ad un embrione di gioco di tennis.
Per spostare la palla da sinistra verso destra, bisogna:

— Stampare la pallina (SHIFT Q)

10 PRINT "●".

Siamo ora pronti a stampare sulla destra della pallina.
— Ritornare sulla pallina e sostituirla con un bianco, poi stampare una nuova pallina

20 PRINT "s̄ Sp ●" x.

— Ciclare sull'istruzione 20: 30 GOTO 20.

Provate il programma formato dalle tre istruzioni, 10, 20, 30 qui sopra.
Ci sono vari difetti:

— bisognerebbe svuotare lo schermo e piazzarsi sulla linea centrale (12);
— la palla ha percorso tutto lo schermo da sinistra verso destra, riparte verso sinistra sulla linea successiva;
— non c'è regolazione di velocità.

La prima obiezione viene facilmente risolta con 5 PRINT "Clr" *b...b*"
Per la terza, bisogna introdurre un intervallo prima di fare il GOTO in 30, ad esempio con l'aiuto del sottoprogramma seguente:

```
                              1000 T = TI
30 GOSUB 1000: GOTO 20        1010 IF TI-T < D GOTO 1010
                              1020 RETURN
```

D serve per regolare l'intervallo nella velocità di spostamento.
Per la seconda obiezione, si può decidere il movimento seguente: andare da sinistra verso destra, poi, quando la pallina arriva all'estremità della linea, andare da destra verso sinistra, ecc.
Per questo, dobbiamo utilizzare una variabile K che tiene il numero della colonna nella quale si trova la pallina. Il senso viene invertito quando K = 40 o l.
Perché la pallina si sposti verso sinistra, si deve scrivere:

40 PRINT "s Sp ss ●"; K = K—1

Ne consegue il programma D.3.

```
5 PRINT"[illegible]"
10 PRINT"●";:D=6:K=1
20 PRINT"[illegible]●";:K=K+1
30 GOSUB 1000:IF K<40 GOTO20
40 PRINT"[illegible]●";:K=K-1
50 GOSUB 1000:IF K>1 GOTO 40
60 GOTO 20
303 GOSUB 1000:IF K<40 GOTO20
1000 T=TI
1010 IF TI-T<D GOTO1010
1020 RETURN
```

Con D = 5, realizzate un vero ipnotizzatore. Con d < 3, la pallina va abbastanza veloce perché si creda di vedere una scia. Con D > 9, gli scatti del movimento sono visibili.

Esercizio 7.7 *Far oscilare il valore di pausa tra 1 e 10. (La pausa varia di 1 ad ogni andata e ritorno).*

Esercizio 7.8 *Far oscillare la pallina dall'alto in basso dello schermo.*

Esercizio 7.9 *Fare andare dalla pallina in diagonale (dall'angolo sinistro all'ultima linea posizione 25 e ritorno).*

Tennis

Siamo pronti per giocare a tennis. In realtà, il nostro gioco sarà rudimentale. Partiamo dalla pallina che oscilla da sinistra verso destra, disponiamo un giocatore a destra e un giocatore a sinistra. Il giocatore di destra non deve lasciare passare dalla pallina la posizione 31 sullo schermo. Per mostrare che recupera la pallina deve premere un tasto. Adotteremo il tasto/. Per il giocatore di sinistra saranno la posizione 10 e il tasto Z. Il programma riconoscerà il tasto mediante un GET.
Peraltro, imponiamo ai giocatori di non premere troppo presto sul loro tasto: se il giocatore di destra preme / prima che la pallina sia in colonna 28, o se il giocatore di sinistra preme Z prima che sia in colonna 13, concede un punto al suo avversario.
Ad ogni punto, viene mostrato il punteggio. Il servizio verrà effettuato premendo 'Shift' e useremo una nuova istruzione: *l'istruzione WAIT.*
Nella sua forma *WAIT A, B* il C64 legge il contenuto dell'indirizzo memoria A, e fa l'AND logico con il valore B (compreso tra 0 e 255).
Se il risultato è 0, il test viene rifatto, il che produce un'attesa. Se il risultato è 1, il C64 passa all'istruzione seguente. Ora, premendo il tasto '*Shift*', la locazione di memoria d'indirizzo 653 prende il valore 1, mentre contiene 0 nel caso opposto. Dunque aspettare una pressione del tasto '*Shift*' si scrive WAIT 653,1. Al momento del servizio, la pallina parte dal centro dello schermo. Il senso del movimento viene tratto a sorte.
Ne consegue il programma D.4. Il listing qui sotto è l'adattamento di un programma di La Commode n.1. Il sottoprogramma vuoto in 300 è previsto per essere sostituito da una routine che produca il rumore del rimbalzo della pallina. Dovreste scriverlo facilmente con l'aiuto della fine del capitolo 6.

```
2 PRINT"■■■■■■■■■■***TENNIS***■■■IL GIOCATORE DI SINISTRA
                                        GIOCA CON IL TASTO ■Z■"
3 PRINT"■IL GIOCATORE DI DESTRA GIOCA CON IL TASTO ■/■"
4 PRINT"■IL SERVIZO SI FA CON SHIFT"
5 PRINT"■PRIMA PREMETE QUANDO LA PALLINA È ARRIVATA
                                          NEL VOSTRO CAMPO"
6 PRINT"PIU 'FORTE' VIENE RIMANDATA LA PALLINA"
7 PRINT"■MA SE PREMETE TROPPO PRESTO,CIÒ DA UN PUNTO
                                     AL VOSTRO AVVERSARIO!"
8 PRINT"■PER SMETTERE FARE STOP CON LA PALLINA VISIBILE
                                     (SHIFT SE NECESSARIO)"
9 PRINT"■■FARE SHIFT PER INIZIARE":WAIT652,1
```

```
10 SG=0:SD=0:G$="■":D$="■":F$="■":S$="▬"
20 PRINT"■■■■■■";G$;SG;F$:PRINTSPC(35);"■";D$;SD;F$:TD=4
                                                    :TG=4
23 IFSG>SDTHENPRINT"    ";S$
26 IFSD>SGTHENPRINTTAB(35);S$
30 PRINT"■■■■■■■";:GOSUB200:GOSUB200:PRINT"■";:GOSUB200
                                    :GOSUB200:PRINT"■■■■"
35 WAIT653,1
40 K=20:PRINTTAB(K-1);"●";:POKE198,0
50 IFRND(1)>.5GOTO130
60 TT=TI
70 GETA$:IFA$="Z"GOTO110
80 IF (TI-TT)<TGGOTO70
90 PRINT"■ ■■■ ●";:K=K-1:IFK>=10GOTO60
100 SD=SD+1:D$="■":G$=F$:GOTO20
110 IFK>=13GOTO100
120 TD=2*(12-K):GOSUB30
130 TT=TI
140 GETA$:IFA$="/"GOTO180
150 IF (TI-TT)<TDGOTO140
160 PRINT"■ ●";:K=K+1:IFK<=31GOTO130
170 SG=SG+1:G$="■":D$=F$:GOTO20
180 IFK<=28GOTO170
190 TG=2*(K-29):GOSUB300:GOTO60
200 PRINTTAB(9);"|  |";TAB(27);" |  |":RETURN
300 RETURN
```

Gli "SPRITES" o OGS

Il C64 — e, in questo, è unico sul mercato — fornisce un altro modo abbastanza semplice per creare disegni animati. Sono gli "sprites" o Oggetti Grafici Spostabili (il termine inglese "sprite" significa spirito o fantasma. Il termine OGS è più tecnico, ma meno spaventoso).
Il C64 può gestire 8 OGS simultaneamente, e ciò, indipendentemente da tutto il resto di ciò che è visualizzato. Gli OGS sono numerati da 0 a 7. Bisogna notare che, se un OGS appare sullo schermo, nello stesso posto di un altro oggetto visualizzato (ad esempio un carattere), esso ha la precedenza, cioè nasconde quell'altro oggetto. Allo stesso modo, se due OGS si scontrano sullo schermo, quello di numero più piccolo ha la precedenza, quindi nasconde l'altro. D'altra parte, uno scontro sullo schermo, tra un OGS e un altro oggetto o un altro OGS, si chiama una collisione: viene individuata dal sistema ed è molto utile per la programmazione di giochi. Non ne parleremo oltre in questo libro.
Gli OGS sono definiti da locazioni di memoria, da registri della cassetta VIC che sono propri di ogni OGS, o da registri condivisi tra gli OGS. Quando un registro è condiviso, il bit n° I riguarda l'OGS n° I.
Un OGS è definito da:

— *La sua forma*. Questa si definisce allo stesso modo che un carattere programmabile, ma in una matrice di 21 linee, 24 colonne, così disposte:

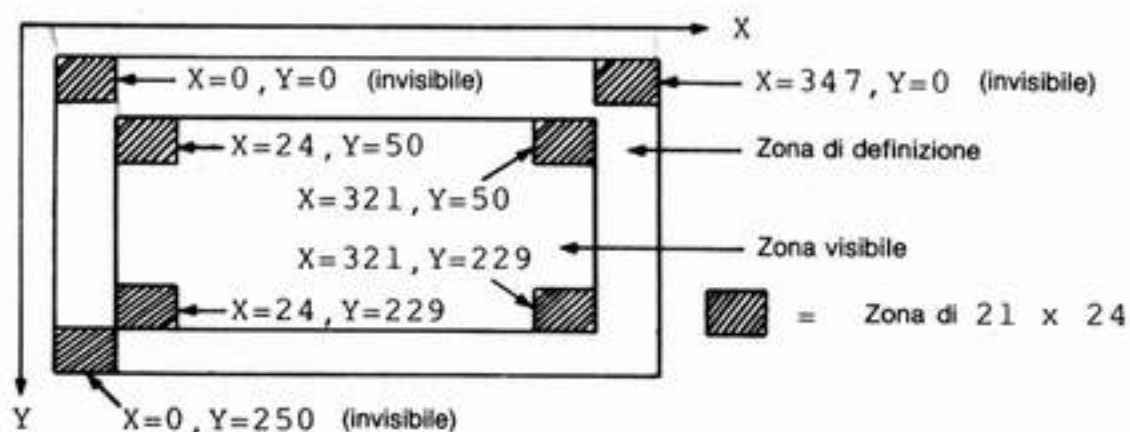
```
             1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
             7 6 5 4 3 2 1 0 7  6  5  4  3  2  1  0  7  6  5  4  3  2  1  0
linea 1          byte 0     |     byte 1            |      byte 2
linea 2          byte 3     |     byte 4            |      byte 5
linea 3          byte 0  .  .  .  .  .  .  .  .  .  .  .  .
                     .                                     .
                     .                                     .
                     .                                     .
linea 21         byte 60    |     byte 61           |      byte 62
```

Ci vogliono quindi 63 bytes per definire ogni OGS. In realtà, viene concesso un 64° byte di riempimento dato che 64 è una potenza di 2. Ciò semplifica l'indirizzamento (nel seguito, si numereranno le linee da 0 a 20 e le colonne da 0 a 23).
Gli indirizzi sono definiti in modo indiretto mediante puntatori in 2040 fino a 2047 (dietro la memoria di schermo).
Il puntatore d'indirizzo 2040+I riguarda l'OGS n° I (I da 0 a 7).
L'indirizzo del byte 0 dell'OGS n° I è 64*PEEK(2040+I).
L'indirizzo del byte K dell'OGS n° I è 64*PEEK(2040+I)+K. Vi si mette il motivo binario che definisce la zona corrispondente dell'OGS.
Si noti che questa definizione indiretta dell'indirizzo permette di definire più OGS della stessa forma: basta che più puntatori siano uguali; punteranno allo stesso indirizzo di memoria e quindi definiranno la stessa forma. Ciò permette pure di tenere in riserva più forme per lo stesso OGS: si passa velocemente da una forma all'altra con un solo POKE nel puntatore.
— *La sua attivazione*. L'OGS n° I viene attivato (appare) se il bit I del byte d'indirizzo 53269 vale 1. Sparisce se tale bit vale 0.

Esempio: fare apparire l'OGS n° I:

```
POKE 53269, PEEK(53269) OR 2↑I
```

— *Il suo modo multicolore*. Non trattato in questo libro: qui, ci limitiamo a degli OGS monocromi.
— *Il suo colore*. Per definire il colore dell'OGS n° I, si fa: POKE 53287+I, codice colore.
— *La sua dilatazione*. Si dilata, in orizzontale, l'OGS n° I con: POKE 53277, PEEK(53277) OR 2↑I lo si dilata, in verticale, con: POKE 53271, PEEK(53271) OR 2↑I.
— *La sua posizione sullo schermo*. È la posizione dell'angolo superiore sinistro quella che viene definita con X compreso tra 0 e 347, e Y compreso tra 0 e 255. Da notare che l'OGS è visibile sullo schermo

soltanto per X compreso tra 50 e 229. Fuori da questo intervallo, l'OGS sparisce nei bordi o anche fuori dello schermo (vedi la figura qui sotto).

Ora specificare Y è facile. Si fa:

POKE 53249+2*I, Y (I = numero dell'OGS)

Per X, è più difficile perché X non sta in un byte. Se X > 255, c'è un 9° bit di valore 1. Per il byte basso di X, c'è un registro per ogni OGS;

POKE 53248+2*I, X AND 255

I 9° bits delle X degli 8 OGS sono, in quanto a loro, raggruppati nel registro 53264:

POKE 53264, (PEEK(53264) AND (255—2↑I) OR (—2↑I)*(X > 255).

Questa espressione ha l'aria complicata, ma non fa altro che sfruttare il fatto che una espressione logica viene valutata, come 0 se è falsa, e —1 se è vera.
Siamo praticamente pronti a far evoluire degli OGS sullo schermo. Ci manca di sapere in quali indirizzi di memoria definiremo i nostri OGS, quindi che valori metteremo in 2040 fino a 2047.
Ci sono due scelte consigliate:
— Se avete soltanto 3 OGS al massimo, potete fare:

POKE 2040,13 (indirizzi dell'OGS 0: 832-895)
POKE 2041,14 (indirizzi dell'OGS 1: 896-859)
POKE 2042,15 (indirizzi dell'OGS 2: 960-1023)

il che equivale a piazzare gli OGS nel tampone cassetta, dunque vale mentre non utilizzate la cassetta.
— Si possono mettere i dati in 12288 e seguenti, cosa che viene fatta con : POKE 2040+I, 192+I

Non dimenticare di proteggere dal Basic la zona 12288... con:

POKE 52,48: POKE 56,48: CLR

Il programma seguente è un "editor di OGS". Vi permette di definire visualmente i vostri OGS, in una griglia di punti sullo schermo: i . saranno dei vuoti, mentre le X rappresenteranno gli elementi accesi dell'OGS. Per accendere una X in una posizione, ci andate con movimenti del cursore sulla linea. Terminate la linea con RETURN.

PROGRAMMA D.5

EDITOR DI OGS

```
60010 POKE52,48:POKE56,48:CLR
60020 FOR I=0 TO 7:POKE 2040+I,192+I:NEXT
60030 INPUT "]NO DELL'OGS DA DEFINIRE";I:IF I>7 THEN END
60040 A=64*PEEK(2040+I)
60050 FOR M=A TO A+63:POKE M,0:NEXT
60060 FOR L=0 TO 20
60070 PRINT"? ........................":NEXT
60080 PRINT"[illegible]";
60090 FOR L=0 TO 20:INPUT A$
60100 FOR K=0 TO 23
60110 J=INT(K/8):B=K AND 7
60120 B$=MID$(A$,K+1,1):M=A+3*L+J:PEEK(M)
60130 IF B$="X" THEN POKE M,C OR 2↑(7-B)
60140 NEXT:NEXT
60150 PRINT"]":POKE 53287+I,0:POKE 53269,PEEK(53269) OR 2↑I
60160 POKE 53249+2*I,100:POKE 53248+2*I,100:POKE53264,0
60170 FOR T=1 TO 5000:NEXT
60180 POKE 53269,PEEK(56249) AND (255-2↑I)
60290 GOTO 60030
```

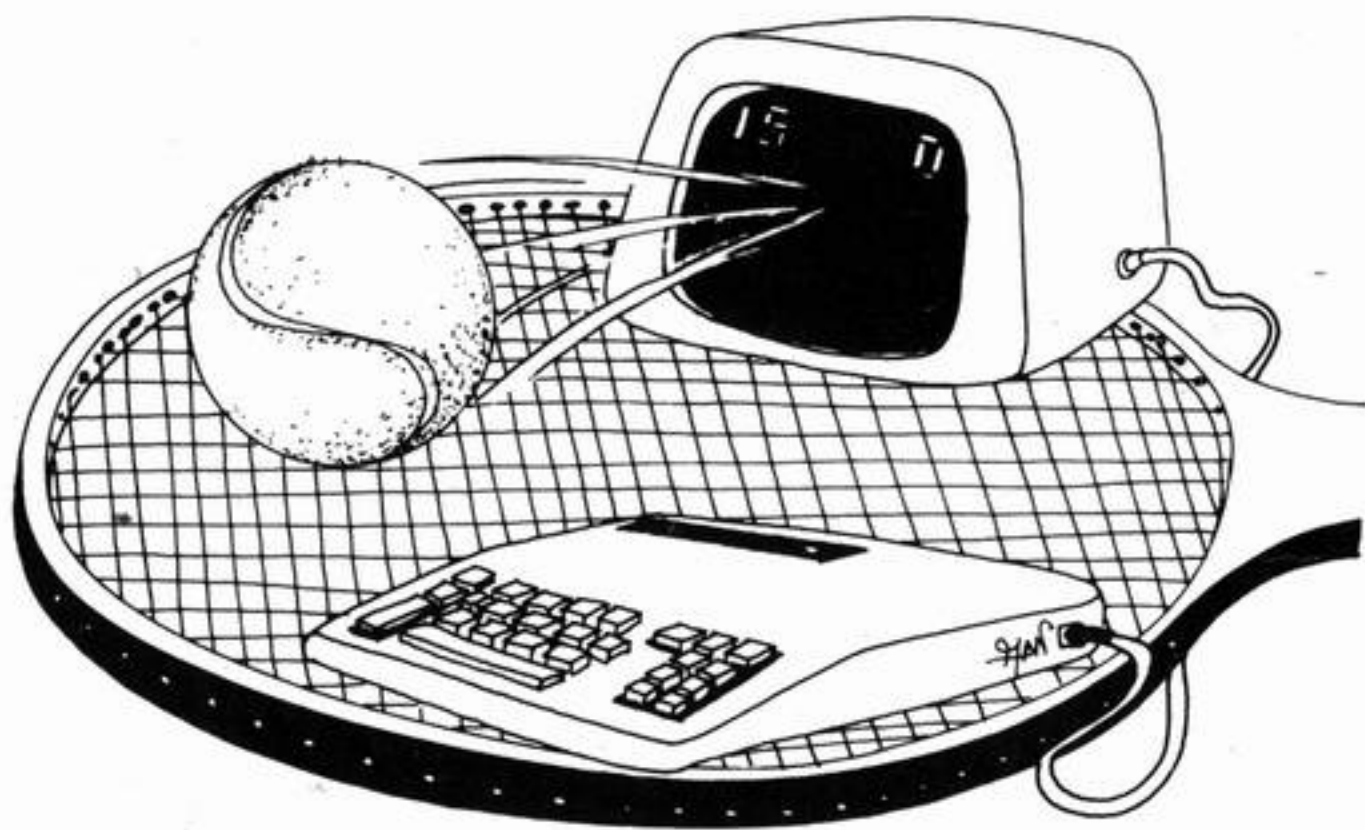
Il programma termina con la visualizzazione dell'OGS che è appena stato creato durante 5 s, poi riprende la creazione di un nuovo OGS (lo fermate dicendo che il numero dell'OGS da creare è ≥ 8; potete ricominciare la creazione dello stesso OGS se non siete soddisfatti, ridando lo stesso numero.
Il programma sta nelle linee di numero 60010 e seguenti. Per poter essere incorporato alla fine del vostro programma di uso, prima fate RUN 60010 per creare i vostri OGS, poi RUN per usarli.

Esercizio 7.10 *Create un OGS. Fatelo spostare da sinistra verso destra, poi da destra verso sinistra, sullo schermo.*

Esercizio 7.11 *Create un OGS. Fatene apparire tre versioni in tre posti diversi dello schermo e in tre colori diversi.*

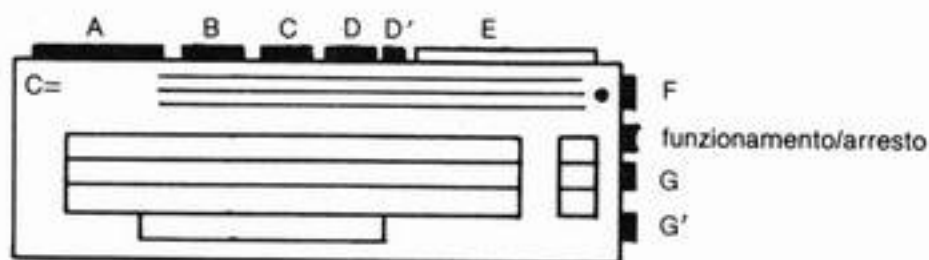
RICAPITOLAZIONE

Abbiamo avuto, ormai, un'idea della gran parte delle tecniche di programmazione accessibili al C64, tanto nel campo dei calcoli quanto in quello dei giochi e dei trattamenti grafici.
Per quanto riguarda i disegni animati, abbiamo visto che il Basic è un po' lento per alcune applicazioni, sebbene gli OGS diano interessanti possibilità.
In tal caso, c'è una risorsa che consiste nel linguaggio macchina del microprocessore del C64.
Le tecniche corrispondenti verranno affrontate in un'altra opera di questa collana.

Capitolo 8

Il C64 e il mondo esterno

Prima di affrontare la conclusione di questo libro, desideriamo darvi un breve resoconto delle caratteristiche del C64 che moltiplicano le già numerose possibilità viste finora. Si tratta delle interfacce del VIC con il mondo esterno.
Infatti, nei capitoli che precedono, abbiamo visto le possibilità interne del C64. Ma il C64 può, inoltre, agire sul mondo esterno e venire influenzato da esso, il che crea molteplici applicazioni.
Qui vedremo soltanto brevi nozioni sulle interfacce del C64; infatti esse vanno al di là del tema di questo libro, e verranno trattate in un prossimo volume di questa serie.

Le interfacce del C64

La figura qui sotto è una veduta da vicino del C64 che mostra i sette connettori mediante i quali esso comunica con il mondo esterno.

A è il connettore "parte interna". Trasmette un certo numero di segnali interni, ma soprattutto un porto di 8 bits paralleli a disposizione dell'utente. È quello che useremo nella sezione seguente. Si deve notare che questi segnali possono anche essere utilizzati per trasformare il vostro C64 in terminale: questo viene chiamato l'interfaccia RS 232 e i programmi di gestione sono incorporati nelle memorie morte del C64.
B è il connettore per il magnetofono a cassette.

C è il connettore di quello che viene chiamato l'interfaccia IEEE serie che permette di collegare la stampa e l'unità dischi del C64. Un'estensione presto disponibile permette di trasformare questa interfaccia in IEEE 488 parallela. Questa interfaccia (parallela) corrisponde ad una norma molto diffusa. Essa permette di collegare le periferiche del resto della gamma Commodore e centinaia di apparecchi di strumentazione (voltmetri, frequenziometri, spettrografi, ecc.) o altre periferiche (esempio: tracciatori di grafici) — (si inserisce nel connettore E).
D e D' sono i connettori video che collegano il C64 ad un televisore. Ricordiamo che potete collegare D' alla presa di antenna del televisore, per il tramite del "modulatore" incorporato nel C64. Ma, a nostro parere, la qualità dell'immagine è migliore se collegate D direttamente alla presa PERITEL del vostro televisore (a condizione che esso ne sia equipaggiato).
E è il "connettore di estensione di memoria". Esso trasporta i segnali del microprocessore del C64. A lui vengono collegate le estensioni di memoria:

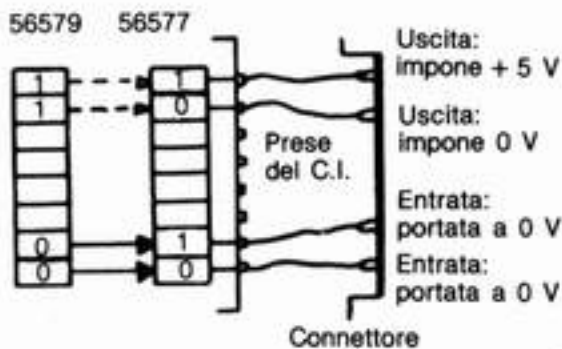
— interfacce o estensioni d'entrata-uscita;
— memorie morte che formano scatole di giochi o che portano funzioni utilitarie (quelle disponibili più velocemente saranno un istruttore linguaggio-macchina e delle estensioni del Basic nel campo dei grafici e del colore).

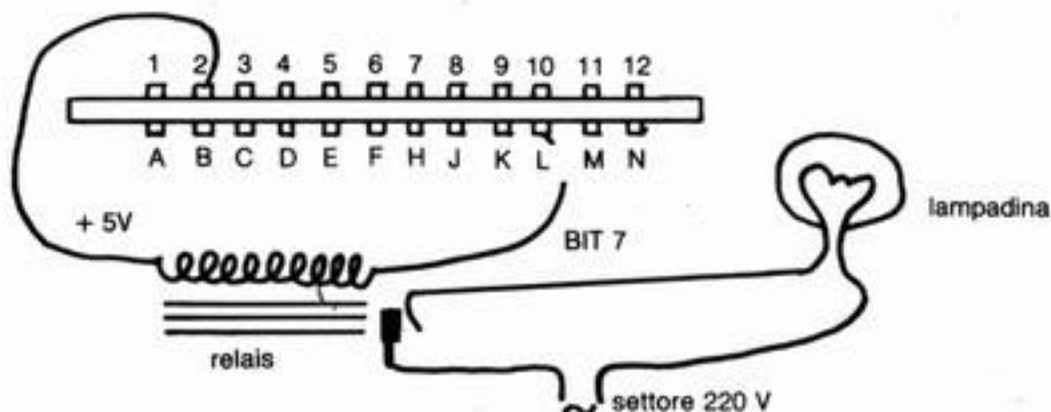
F è il connettore di alimentazione al quale collegate il trasformatore, collegato lui stesso alla presa di settore.
G e G' sono i connettori di giochi ai quali potete collegare manopole di giochi, potenziometri, una matita luminosa, ecc.

Accensione di una lampadina

Come esempio rudimentale del modo in cui il C64 può agire sul mondo esterno, descriveremo come può comandare l'accensione o lo spegnimento di una lampadina.
Siccome — ricordate — il C64 conosce l'ora grazie al suo orologio tempo reale, è possibile chiedere l'accensione o l'estinzione ad ore stabilite.
Ciò permette applicazioni come lo spegnimento di tutti gli uffici dopo le 18.
Risponderete che un programmatore a contatti ha lo stesso effetto; non è del tutto vero: il C64 può tenere conto dei giorni lavorativi, delle feste mobili e pure degli anni bisestili. Una tale applicazione ammortizza il prezzo del C64 in 4 mesi, tanto più che si possono nello stesso tempo gestire altri risparmi di energia.
La base di questa applicazione è il PIA. Studiamolo ora in modo succinto.

Nozione di PIA

(Peripheral Interface Adaptor = adattatore di interfaccia periferica).
Il PIA è un circuito integrato annesso del microprocessore.
Un PIA contiene generalmente due "porti".
Un porto di PIA è visto dal microprocessore come una locazione di memoria di 8 bit, che ha un indirizzo nel quale si può fare PEEK o POKE. Nel nostro caso, questo indirizzo è 56577.
Ma questa "locazione di memoria" ha un comportamento particolare: ognuno dei suoi 8 bit è collegato ad una presa esterna del circuito integrato, e, più in là, ad un morsetto del connettore C.

(Cf figura qui di fronte). Se scriviamo un 1 con POKE su di un bit dell'indirizzo 56577, la tensione al morsetto corrispondente verrà portata a +5 V, varrà 0 se scriviamo uno 0.

Allo stesso modo se leggiamo l'indirizzo 56577 (con PEEK) otterremo un 1 se il morsetto è portato a 5 V dall'esterno, uno 0 se è portato a 0 V.
Prima di tutto bisogna decidere se il morsetto imporrà il suo stato all'esterno (si dice che è in uscita), o se, all'inverso, si metterà all'"ascolto" del mondo esterno (si dice che è in entrata).
Viene usata un'altra locazione di memoria (56579) ogni bit della quale corrisponde ad un bit della locazione (56577): se c'è un 1 in un bit di 56579, il corrispondente bit di 56577 è in uscita; è in entrata se c'è uno 0.
Creiamo ora il collegamento della figura qui sotto:

Connettore A visto dal dietro del C64

Si collega il bit 7 del porto (56577) ad un "relais" a bassa intensità, riportato al +5 V preso dallo stesso connettore (presa 2). Il bit viene messo in uscita scrivendo 1 nel bit 7 di 56579, il che viene fatto con POKE 56579,128 (infatti $2^7 = 128$).
Se ora scriviamo un 1 sul bit 7 di 56577, avremo 5 V, quindi 5 V da ogni lato del "relais": quindi il contatto sarà aperto, la lampadina sarà spenta.
Se, al contrario quel bit viene messo a zero, la tensione sarà nulla, passerà della corrente nella bobina del "relais", il contatto sarà chiuso, la lampadina si accenderà.
Il programma E.1 è un esempio di uno di questi fenomeni:

PROGRAMMA E.1

```
10 INPUT "A CHE ORA DEVO ACCENDERE";TA$
20 INPUT "A CHE ORA DEVO SPEGNERE";TE$
30 INPUT "CHE ORE SONO";TI$
40 POKE 56579,128:POKE 56577,128
50 IF TI$<TA$ GOTO 50
60 POKE 56577,0
70 IF TI$<TE$ GOTO 70
80 POKE 56577,128
```

Questo programma fornisce soltanto lo scheletro dell'applicazione economia di luce che progettavamo qui sopra. Tuttavia si può notare l'estrema semplicità di questa prima applicazione.

Esercizio 8.1 *Collegare un altoparlante al posto del "relais" ed eseguire il programma:*

10 POKE 56579,128
20 POKE 56577,128: POKE 56577,0: GOTO 20

Che cosa succede?

Domanda: avete detto che un PIA ha due porti di 8 bit. Uno solo va sul connettore A. Cosa ne è del secondo?

Viene utilizzato internamente dal C64, il quale ha d'altra parte diversi PIA che utilizza per comandare le sue varie periferiche.

Eccoci giunti al termine di questo libro, al termine della nostra scoperta del C64.
Bisogna ammettere che si tratta di una macchina meravigliosa, soprattutto in rapporto al suo prezzo.
Le sue possibilità sono immense, ne abbiamo scoperto soltanto una piccola parte, tanto nei calcoli quanto in gestione, come macchina di

svago e di educazione: il nostro programma di quiz geografico vi permette di rivedere la geografia divertendovi... Le possibilità di trattamento grafico sono particolarmente utili in questo contesto, e particolarmente perfezionate sul C64.
Infine, abbiamo accennato come il C64 può agire sul suo ambiente, e come può essere esteso per avere ancora più possibilità.
Lasciamo ora il campo alla vostra immaginazione perché ne tragga il miglior profitto.

Appendice 1

Funzioni e parole chiave del Basic

FUNZIONI ARITMETICHE

ABS Valore assoluto dell'argomento tra parentesi.

ATN Arco tangente — il risultato è in radianti, compreso tra $-\pi/2$ e $\pi/2$.

COS Coseno — l'argomento deve essere in radianti
Esempio: cos(X in gradi) = $\cos(\pi * X/180)$.

EXP Esponenziale e^x. L'argomento deve essere < 88, altrimenti si produce un superamento di capacità.

FRE Per qualsiasi valore dell'argomento, fornisce il numero di byte rimasti liberi in memoria. Esempio: subito dopo l'accensione, ?FRE(0) fornisce —26627. Bisogna chiedere ?65536+FRE(0) (è un complemento, vedi pagina 133). Otteniamo 38909 e non 38911 poiché il comando occupa due byte.

INT Parte intera, o piuttosto numero intero più grande, minore o uguale all'argomento: INT(0.5) vale 0; INT(5) vale 5,INT(—0.5) vale —1, INT(—3) vale —3.

LOG Logaritmo naturale (neperiano o in base e). Per ottenere il logaritmo di X in base Y, usare LOG(X)/LOG(Y). Esempio: logaritmo decimale di X = LOG(X)/LOG(10).

PEEK Fornisce il contenuto (compreso tra 0 e 255) della locazione di memoria il cui indirizzo è uguale al suo argomento (che deve essere intero e compreso tra 0 e 65535). Per scrivere in memoria, vedi POKE, pagina 127.

POS POS(0) fornisce la prossima posizione di stampa libera sulla linea di schermo (posizione del cursore).

RND Fornisce un numero pseudocasuale compreso tra 0 e 1. Vedi la spiegazione dettagliata al capitolo 5.

SGN Fornisce "segno": 1 se $X > 0$, -1 se $X < 0$ e 0 se $X = 0$.

SIN Seno — l'argomento è supposto in radianti.

SPC Può usarsi soltanto in una istruzione PRINT. PRINT SPC(I) stampa I spazi. I deve essere intero e compreso tra 0 e 255.

SQR Radice Quadrata. L'argomento deve essere maggiore o uguale a 0.

TAB Può usarsi soltanto in una istruzione PRINT. PRINT TAB(I); fa andare alla posizione di stampa n° I (0 è la posizione più a sinistra di una linea, 39 la più a destra). I deve essere compreso tra 0 e 255. Se I < posizione dove si sta già, non c'è alcun effetto, cioè la prossima stampa avverrà nel luogo in cui si era posizionati.

TAN Tangente — l'argomento è supposto in radianti.

USR Richiamo di un programma utente in linguaggio macchina (con trasmissione di un argomento, diversamente da SYS).

FUNZIONI STRINGA

Le funzioni che riguardano le stringhe di caratteri sono descritte nel capitolo 5.
Diamo qui sotto la loro lista:

LEN (X$)	Lunghezza
LEFT$ (X$, N)	Estrazione a sinistra
RIGHT$(X$, N)	Estrazione a destra
MID$ (X$, K)	o **MID$ (X$, K, N)** estrazione in mezzo
ASC (X$)	Conversione di carattere in ASCII
CHR$ (K)	Conversione di ASCII in carattere
STR$(A)	Rappresentazione di un numero
VAL(X$)	Valore rappresentato da una stringa.

PAROLE CHIAVE RISERVATE IN BASIC (e ABBREVIAZIONI)

ABS (Ab)[1]
AND (An)
ASC (As)
ATN (At)
CHR$ (Ch)
CLOSE (CLo)
CLR (Cl)
CMD (Cm)
CONT (Co)
COS

DATA (Da)
DEF (De)
DIM (Di)
END (En)
EXP (Ex)
FN
FOR (Fo)

FRE (Fr)
GET (Ge)
GOSUB (GOs)
GOTO (Go)
IF
INPUT
INPUT#(In)
INT
LEFT$ (LEf)
LEN
LET (le)
LIST (Li)
LOAD (Lo)
LOG
MID$ (Mi)
NEW
NEXT (Ne)
NOT (No)

ON
OPEN (Op)
OR
PEEK (Pe)
POKE (Po)
POS
PRINT (?)
PRINT#(Pr)
READ (Re)
REM

RESTORE (REs)
RETURN (REt)
RIGHT$ (Ri)
RND (Rn)
RUN (Ru)
SAVE (Sa)
SGN (Sg)

SIN (Si)
SPC (Sp)
SQR (Sq)
STEP (STe)
STOP (St)
STR$ (STr)
SYS (Sy)
TAB (Ta)
TAN
THEN (Tn)

TO
USR (Us)
VAL (Va)
VERIFY (Ve)
WAIT (Wa)

(1) Nelle abbreviazioni, le minuscole rappresentano le lettere con SHIFT. Esse appaiono in forma grafica sullo schermo.

Appendice 2

Repertorio delle istruzioni e degli operatori Basic

Come la guida Michelin, assegnamo stellette alle istruzioni e ai comandi: * * * istruzione fondamentale, * * istruzione importante, * istruzione interessante. Nessuna *: istruzione che non bisogna esitare ad utilizzare se se ne sente il bisogno; alcune di queste istruzioni non vengono trattate in questo libro.
Inoltre, P vuol dire: istruzione piuttosto da modo programmato, P̸ vuol dire: proibito in modo programmato, C: comando piuttosto da modo diretto, ₵: proibito in modo diretto, "nulla" vuol dire: ora l'uno ora l'altro.
La definizione è seguita da esempi non commentati che mostrano le varie forme che può assumere l'istruzione.

Categoria	Parola chiave	Definizione-*esempio*	Pag.
	CLOSE	Chiusura di file CLOSE 5	136
C	CLR	Azzeramento di tutte le variabili	21
	CMD	Comando di periferiche	137
CP̸	CONT	Continuare nel programma dopo interruzione	14 48
P*	DATA	Definisce una lista di costanti che verrano "lette" da un'istruzione READ. *10 DATA ABC, DEF, 5,3,25*	
P*	DEF FN	Definizione di una funzione utente. *10 DEF FN F(X)=A*X+B*	96

segue

seguito

Categoria	Parola chiave	Definizione-*esempio*	Pag.
P*	DIM	Dimensionamento di un array (fissa i valori massimi degli indici). *10 DIM A(15), B(3), C$(2,5), D(7,6,2)-20 DIM W(N)*	53
P₵*	END	Termina un programma	22
**	FOR	Introduce un ciclo: tutte le istruzioni comprese tra FOR I = A TO B STEP C e il NEXT corrispondente verranno ripetute per tutti i valori di I da A a B, C per C: *10 FOR I = 1 TO 1000* *50 FOR I = 0 TO 200 STEP 5* *60 FOR I = N TO 3*N + 4 STEP 5* *70 FOR I = 50 TO 1 SYEP —1* *80 FOR I = 1.5 TO 2*Π STEP .1*	41
P₵*	GET	Prende un carattere dalla tastiera *10 GET A$*	74
P**	GOSUB	Richiamo di sottoprogramma *10 GOSUB 1000*	80
P**	GOTO	Salto ad un'altra istruzione *10 GOTO 50*	13
P**	IF	Salto condizionato della forma IF condizione THEN istruzione. Se la condizione non è soddisfatta, si passa alla linea successiva; se la condizione è soddisfatta, si effettua l'istruzione che segue THEN. IF c THEN GOTO x si abbrevia in IF c THEN x o in IF c GOTO x. *10 IF A > B THEN Y = K* *20 IF A = 3 GOTO 1000* *30 IF A$ < >"" THEN 50*	35
P₵***	INPUT	Acquisizione di dati alla tastiera *10 INPUT A* *20 INPUT A, B, C$, D* *30 INPUT "INSERITE UN NUMERO"; N*	11
C***	LIST	Lista del programma *LIST LIST 10— LIST—100 LIST 10 LIST 10—100*	19
C**	LOAD	Caricamento di un programma su cassetta *LOAD LOAD "PROG"*	32
C₣*	NEW	Svuota la memoria programma	20
**	NEXT	Fa andare alla successiva iterazione in un FOR *NEXT I NEXT J,I NEXT*	41
P	ON	*ON I GOTO 10,20,30* Se I vale 1, si va in 10, se vale 2, si va in 20, in 30 se vale 3 *ON I GOSUB 1000, 1500, 2000, 5000* Se I vale 1, si richiama il sottoprogramma in 1000, 2 in 1500, 3 in 2000, 5 in 5000	

segue

seguito

Categoria	Parola chiave	Definizione-*esempio*	Pag.
	OPEN	Apertura di un file *OPEN 1,4 ← stampante* *OPEN 5,1,0,"TATA"* *OPEN L,P*	136
*	POKE	POKE a,b scrive il dato b all'indirizzo a. *POKE 36879,27 POKE K+1,Z—4* *Per leggere in memoria, vedi PEEK*	71
* * *	PRINT	Stampa un risultato sullo schermo. *PRINT A 10 PRINT A; B, J* *20 PRINT2*A+3,B$* *30 PRINT "IL RISULTATO E"; B;*	11
P*	READ	"Lettura" di dati in una istruzione DATA associata. *10 READ A 20 READ A, B$, C*	51
P	REM	Introduce un commento	56
P	RESTORE	Torna all'inizio dei DATA	51
P* *	RETURN	Ritorno da sottoprogramma. *100 RETURN*	83
C* * *	RUN	Dà inizio all'esecuzione di un programma. *RUN RUN 500*	6 21
C* *	SAVE	Registra un programma su cassetta. *SAVE SAVE "TATA"*	30
* *	STEP	Introduce il passo di incremento di un FOR	44
P₵	STOP	Ferma l'esecuzione di un programma *10 STOP*	48
	SYS	Fa partire l'esecuzione di un programma in linguaggio macchina all'indirizzo indicato. *25 SYS 64738*	
P* *	THEN	Introduce l'istruzione da effettuare quando un IF viene soddisfatto	35
* *	TO	Introduce il valore limite di un FOR	43
C₱	VERIFY	Verifica di un programma che è appena stato registrato su cassetta	31
	WAIT	*WAIT A,B,C,* (C=0 se assente) Sospende l'esecuzione del programma fino a che (contenuto dell'indirizzo A) AND (B) OR esclusivo (C) sia uguale ad 1. *WAIT 653,1* attende che si prema il tasto 'Shift'.	108
* * *	=	Assegnamento di valore ad una variabile.	11

OPERATORI

Aritmetici

- \+ addizione di numeri o concatenazione di stringhe di caratteri
- — prendere l'opposto o sottrazione
- * moltiplicazione
- / divisione

di rilassamento

=uguale	< >diverso
<minore	>maggiore
< =min. o =	> =mag. o =
= <min. o =	= >mag. o =

Logici effettuati bit per bit

NOT non logico, agisce su un operando solo

AND e logico } 2 operandi
OR o logico }

bit 1	bit 2	AND	OR
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

bit	NOT
0	1
1	0

Appendice 3

Messaggi d'errore

Il fatto di disporre di un interprete che prende le istruzioni Basic una per una per eseguirle permette, quando si verifica un errore, di identificare, con precisione, l'istruzione nella quale si è prodotto l'incidente.
In conseguenza, i messaggi di errore saranno della forma:

?messaggio ERROR IN numero

precisando il numero della linea nella quale l'errore è stato incontrato. L'unico caso in cui non viene precisato il numero è quello di un comando diretto.
I problemi più correnti provengono da una parola chiave ortografata male, da una virgola mancante o da una variabile di un certo tipo usata a torto, ecc.
Dobbiamo notare che un errore segnalato ad un numero di linea può non essere altro che la conseguenza di un altro errore: ad esempio un DIVISION BY ZERO ERROR proviene piuttosto dalla istruzione precedente dove viene calcolato il divisore.
Diamo, qui sotto, la traduzione e, per i più importanti, spiegazioni di diversi messaggi di errore che possono essere mandati dal C64 al momento dell'esecuzione di un programma Basic.

BAD SUBSCRIPT (cattivo indice)

Tentativo di chiamare un elemento di array d'indice superiore al limite fornito nel DIM.

Esempio: DIM A(15) con A(20) o A(2,2).

BREAK ERROR (arresto)

È stato premuto il tasto Stop, mentre si stava eseguendo un comando su cassetta.

CAN'T CONTINUE (non si può continuare)

Impossibilità di riprendere l'esecuzione di un programma con CONT. Succede se c'è stato un errore, o se, durante l'interruzione è stata modificata o aggiunta un'istruzione. In certi casi si può riprendere con GOTO numero.

DIVISION BY ZERO (divisione per zero)

Proviene il più delle volte da una variabile non inizializzata.

FORMULA TOO COMPLEX (espressione stringa di caratteri troppo complessa)

ILLEGAL DIRECT (illegale in modo diretto)

INPUT, GET e DEF FN sono proibite in modo diretto.

ILLEGAL QUANTITY (valore errato)

Uso di una funzione con un argomento all'infuori dell'intervallo permesso. Esempi:

— indice negativo o > 32767;
— argomento di LOG negativo o nullo;
— argomento di SQR negativo;
— 0 potenza numero negativo;
— lunghezza specificata in MID$, RIGHT$, LEFT$ non compresa tra 0 e 255;
— indice di ON GOTO fuori dei limiti;
— indirizzi in PEEK, POKE, WAIT o SYS non compresi tra 0 e 65535;
— byte specificato in WAIT, POKE, TAB o SPC non compreso tra 0 e 255.

NEXT WITHOUT FOR (NEXT senza FOR corrispondente)

Proviene il più delle volte da cicli mal orientati, da una confusione sulla variabile segnata nel NEXT oppure se si è soppresso un FOR per una correzione, dimenticando di sopprimere il NEXT.

OUT OF DATA (DATA esaurito)

Si prova a fare un READ mentre sono già stati "letti" tutti i dati di tutte le DATA. Bisogna, o contare con cura i dati, o usare RESTORE. Si noti che questo messaggio appare se facciamo *Return*' sulla linea del messaggio READY che viene interpretato come un READ Y.

OUT OF MEMORY (memoria esaurita)

Programma troppo lungo, o troppe variabili, o troppi cicli e GOSUB indentati.

OVERFLOW (superamento di capacità)

Risultato di un calcolo superiore a 1.7E38. Nell'altro senso, un risultato inferiore a 2.9E—39 è indiscernibile da 0, ma non c'è messaggio d'errore.

REDIM'D ARRAY (array ridimensionato)

Siamo passati una seconda volta sull'istruzione DIM riguardo ad un array, oppure c'è una seconda istruzione DIM per un array.

REDO FROM START (ricomincia dall'inizio)

Al momento di un'istruzione INPUT, abbiamo fornito una quantità alfanumerica, quando ci si aspettava un dato numerico. Bisogna ricominciare ridando tutti i valori attesi dall'istruzione INPUT considerato.

RETURN WITHOUT GOSUB (ritorno senza GOSUB)

Siamo capitati su un RETURN mentre non era stato appena eseguito un GOSUB. Il più delle volte dovuto al fatto che si è dimenticato END alla fine del programma principale che precede un sotto-programma.

STRING TOO LONG (stringa di caratteri troppo lunga)

Tentativo di creare (per concatenamento) una stringa di più di 255 caratteri.

SYNTAX (errore di sintassi)

Istruzione incomprensibile per Basic. Dovuto soprattutto a parentesi non accoppiate, caratteri illegali, cattiva punteggiatura, errore di ortografia in una parola chiave.

TYPE MISMATCH (disaccordo tra numerico e alfanumerico)

Valore numerico assegnato ad una variabile stringa, o viceversa, oppure argomento numerico fornito ad una funzione che richiede un argomento alfanumerico, o viceversa.

UNDEF'D STATEMENT (istruzione non definita)

GOTO, GOSUB o THEN che rimandano ad un numero di linea inesistente. Il più delle volte è dovuto al fatto che si è dimenticato di correggere un GOTO... quando la linea puntata è stata soppressa o spostata.

UNDEF'D FUNCTION (funzione non definita)

Riferimento ad una funzione per la quale è stato dimenticato il DEF FN.

Non parleremo, qui, dei messaggi che riguardano i files, poiché tale questione non viene trattata in questo libro. In quanto a LOAD e VERIFY ERROR che riguardano la registrazione dei programmi su cassetta essi vengono trattati nel capitolo 3.

Appendice 4

Domande e risposte

Perché, per sapere che grandezza di memoria rimane libera, bisogna fare ?65536+FRE(0) e non semplicemente ?FRE(0)?

Perché il vostro C64 ha troppa memoria. La funzione FRE è stata scritta per macchine più piccole: dà quindi un risultato su 16 bits, compreso tra —32768 e +32767, il che bastava per le piccole memorie. Ora, se la memoria rimasta libera supera 32767 bytes, ottenete il risultato in complemento a due ed appare negativo. Ristabilite tutto sommando 65536. Potete trovare dettagli sui complementi a due e sul funzionamento dei microcomputers in DJ. DAVID "Les Systèmes à Microprocesseur" — EDITESTS.

Quando voglio lanciare un programma dopo un'interruzione (in 100), posso usare CONT, RUN 100 o GOTO 100 (in modo diretto). Qual è la differenza?

CONT non può essere usato se, nel corso dell'istruzione, avete apportato una correzione al programma. Se usate RUN 100, tutte le vostre variabili verranno azzerate. Dunque se al momento dell'interruzione avete corretto una variabile bisogna usare GOTO 100.

Quando la variabile da leggere con INPUT è una stringa di caratteri, la si deve mettere tra apici?

Normalmente no. La si mette tra apici se la stringa contiene movimenti del cursore, o spazi all'inizio o alla fine.

Che cosa succcede quando in INPUT di un numero forniamo una stringa di caratteri o viceversa?

Se ad INPUT A$ rispondete 123, il C64 prenderà la stringa di caratteri cifra 1, cifra 2, cifra 3; nessun problema. Se ad INPUT A rispondete ABC, ci sarà un errore con il messaggio REDO FROM START e l'istru-

zione INPUT verrà rieseguita; quindi, per un INPUT a più variabili, dovrete ridare tutti i dati.

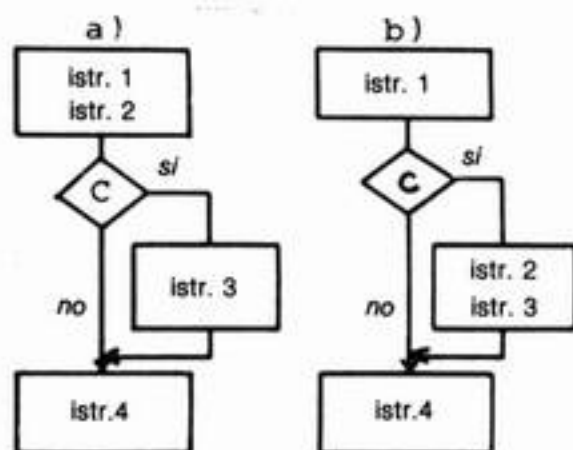
C'è un limite alla lunghezza delle stringhe di caratteri?

Sì: 255 caratteri, che è già molto. Si noti che una stringa tale non può essere data in un colpo solo, poiché una istruzione non può superare 80 caratteri; essa deve essere costruita per concatenamento.

Come prevedere la grandezza di un programma?

Approssimativamente, un programma occupa tanti bytes quanti caratteri racchiude, poiché viene caricato tale quale, come stringa di caratteri tranne le parole chiave che vengono sostituite con un codice in 1 byte. Per quanto riguarda le variabili, ogni variabile numerica occupa 7 bytes, ogni stringa occupa (7+lunghezza) bytes. Un array occupa $x(n+1)+2d$, dove d è il numero di dimensioni, n la grandezza dell'array (compreso l'elemento n° 0) e $x=5$ (numeri) o 3 (stringhe di caratteri). Si guadagna del posto memoria sopprimendo tutti i bianchi inutili, mettendo più istruzioni per linea, evitando i REM, usando variabili piuttosto che costanti. Usate dei GOSUB non appena bisogna usare più volte una sequenza di istruzioni identiche.

Ho dei problemi quando ho più istruzioni sulla stessa linea, con un IF tra di esse.

La forma: 10 istr. 1: istr. 2: IF c THEN istr. 3:
20 istr. 4
non vi deve causare alcun problema. Obbedisce allo schema a blocchi a) qui sotto;

La forma:
5 istr. 1
10 IF c THEN istr. 2: istr. 3
20 istr. 4

obbedisce, **sul C64** *allo schema a blocchi b) qui di fronte. Cioè quando la condizione non è soddisfatta, si passa alla linea successiva e non all'istruzione che segue l'istruzione introdotta da THEN.*

Cosa succede se metto un'istruzione FOR senza NEXT?

Le istruzioni che seguono FOR vengono eseguite una volta (poiché nulla dice al C64 che bisogna ricominciare). La variabile, citata nel FOR, prende per valore il valore di partenza (messo prima di TO).

Ho dei problemi con i caratteri movimento del cursore quando correggo un programma.

Tutti i problemi provengono del fatto che, quando si vuole un movimento effettivo del cursore, il sistema si trova in modo apici e quindi stampa un carattere movimento del cursore differito (s̄, d̲ ecc.) o viceversa. Ecco due casi tipici:
— Quando è stato appena battuto Īnst, ci si trova in modo apici nella finestra di inserzione quindi, se si prova a metterci dei caratteri cursore, si otterranno i caratteri differiti.
— Quando si entra in una zona tra apici, con movimenti del cursore nel corso di una correzione, non si è in modo apici. Qui, ad esempio, in 10 PRINT "āxxx" si vuole sostituire ā (listato ■) con b (listato ■), bisogna portare il cursore sul ■ , fare Inst, b̲, d̲d̲ per portare il cursore sulla prima x, e finalmente fare Dēl poi Return. Un altro modo sarebbe stato di portare il cursore sull'apice aperto e di ribattere l'apice.
In tutti i casi in cui stiamo in modo apici mentre non lo desideriamo, se ne esce facendo Return e tornando sull'istruzione.

Come stampare numeri allineati sulla destra?

Si devono utilizzare le funzioni stringhe di caratteri. Ad esempio, disponiamo di una zona di 10 caratteri nella quale vogliamo stampare un numero N. Quale che sia il numero di cifre, vogliamo che il numero venga stampato a destra della zona. Usiamo la sequenza seguente;

```
100 N$ = STR$(N):L = LEN(N$)
110 IF L = 10 GOTO 130
120 FOR I = 1 To 10—L:N$ = " " +N$:NEXT
130 PRINT N$
```

possiamo sostituire 120 con 120 PRINT SPC(10—L); possiamo pure sostituire 110 e 120 con N$ = RIGHT$(" 10 sp." +N$, 10)

Ho appena fatto un programma che vorrei salvare su cassetta, ma ho dimenticato di posizionare la cassetta e pure di segnare il valore del contatore.

Soprattutto non dovreste fare SAVE con la cassetta riavvolta: cancellereste una parte dei programmi già registrati. Quello che bisogna fare, è posizionarsi dietro all'ultimo programma che era stato registrato

su cassetta; supponiamo che si chiami ZOZO. Non possiamo fare un LOAD "ZOZO" perché ZOZO verrebbe in memoria a cancellare il programma che avete appena fatto. La soluzione è di fare VERIFY "ZOZO". Il contenuto di ZOZO verrà confrontato con il contenuto della memoria che non verrà alterato. Avremo un messaggio ?VERIFY ERROR, che è normale e al quale non bisogna fare attenzione: la cassetta verrà posizionata dietro ZOZO. Potete quindi salvare il vostro programma.

Come sapere che cosa c'è su di una cassetta?

È consigliabile tenere un repertorio delle cassette via via che si riempiono. Ma se non l'avete fatto, basta fare un VERIFY "BZZBZZ". Si suppone che BZZBZZ non sia il nome di nessuno dei programmi sulla cassetta. Allora, il C64 percorrerà tutta la cassetta, e per ogni programma che troverà, stamperà: FOUND nome. E siccome nome è diverso da BZZBZZ andrà avanti.

Ho visto, su certi listings, nomi di variabili che terminano con %. Che cos'è?

Si tratta di un tipo di variabile che non abbiamo studiato: le variabili intere. I loro nomi sono della forma A% A1% ALBERTO%.
Possiamo, in un programma, avere le tre variabili distinte A% A e A$.
Le variabili % possono essere dimensionate. Queste variabili possono soltanto prendere valori interi compresi tra —32767 e +32767. Sono interessanti per economizzare memoria, dato che occupano soltanto due bytes.

A parte programmi, si possono scrivere anche dati su cassetta?

Sì. Benché questa questione faccia parte dell'insieme dei trattamenti di files, superi l'oggetto di questo libro e venga trattata in un'altra opera, vi diamo alcuni elementi:
Per scrivere dei dati:
— dobbiamo prima di tutto aprire un file OPEN 5,1,1, "NOME".

numero logico (5); periferica (1); scrittura (1); nome di file gestito come un nome di programma ("NOME").

I numeri di periferiche usuali sono:
1 = cassetta, 2 = interfaccia RS232; 4 = stampante; 8 = disco.
— Poi, dare gli ordini di scritture successive dei dati:
PRINT# 5,A ← variabile da scrivere
stesso numero logico.
— Infine, chiudere il file: CLOSE 5
Per leggere:
— apertura OPEN 5,1,0 "NOME" ←(permette di ritrovare il file);
0: lettura

— ordini di letture successive INPUT #5,A (analogo all'INPUT tastiera);
— chiusura CLOSE 5.

Ho una stampante. Come usarla?

Anche qui, affrontiamo problemi di files. Ci limiteremo quindi ad informazioni scheletriche.
Due modi di usare la vostra stampante:

1° Modo analogo ai files di dati su cassetta:
OPEN 1,4 Nessun nome, sappiamo che è in scrittura
numero stampante
logico

PRINT # 1,A
CLOSE 1

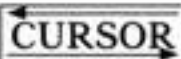
2° Se fate all'inizio del programma o, in modo diretto prima del GOTO di lancio del programma:
OPEN 1,4
e CMD 1
allora ogni vostro ordine PRINT schermo andrà sulla stampante, come se ci fosse stato PRINT #1
Allo stesso modo, i LIST vanno sulla stampante quindi, per ottenere un LISTING su stampante, battete in modo diretto: OPEN 1,4: CMD 1; LIST. Si termina con CLOSE 1.

Appendice 5

Soluzione degli esercizi

Gli esercizi non presentati qui, hanno normalmente la soluzione nel testo.

Esercizio 2.1

```
10 INPUT"RAGGIO,ALTEZZA";R,H
20 V=π*R↑2*H
30 PRINT"VOLUME",V
40 GOTO10
100 REM 20 E 30 POTREBBERO ESSERE SOSTITUITI
110 REM CON 25 PRINT "VOLUME=",π*R↑2*H
```

Esercizio 2.2

```
10 INPUT"CAPITALE,TASSO IN %,NUMERO DI ANNI";C,T,N
20 I=C*((1+T/100)↑N-1)
30 PRINT"INTERESSE=";I
40 GOTO10
```

Esercizio 3.2 Premete "Shift" (tenendolo premuto) poi CURSOR quattro volte. *'Inst/Del'* (due volte) sempre con 'Shift', lasciate *'Shift'* e battete GI.

Esercizio 3.4 Affinché l'istruzione 50 passi da 30 PRINT "SUPERFICIE=", S a 30 PRINT "VOLUME=", V; portare il cursore sulla S di superficie e battere: VOLUME CURSOR cancellare ICIE con 'Inst/Del' quattro volte poi CURSOR fino alla S che sostituite con V.

Esercizio 4.1

```
           15 IF SC=0 GOTO 10
variante:  15 IF SC=0 GOTO 50
           50 PRINT "LA SUPERFICIE DEVE ESSERE POSITIVA"
           60 GOTO 10
```

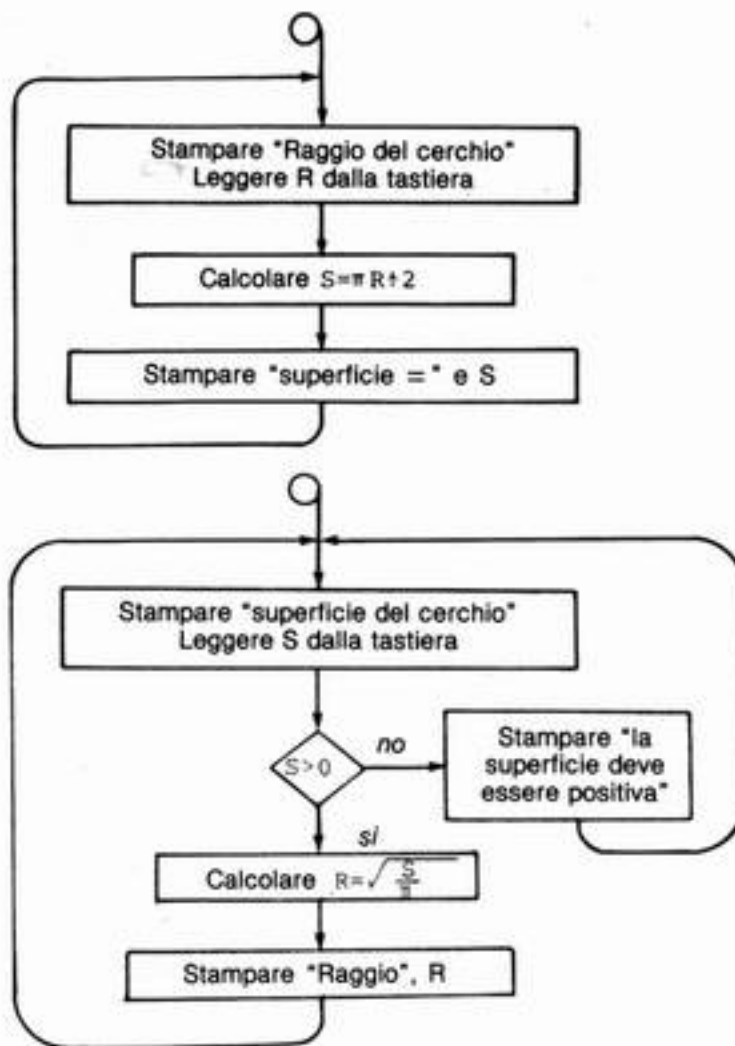
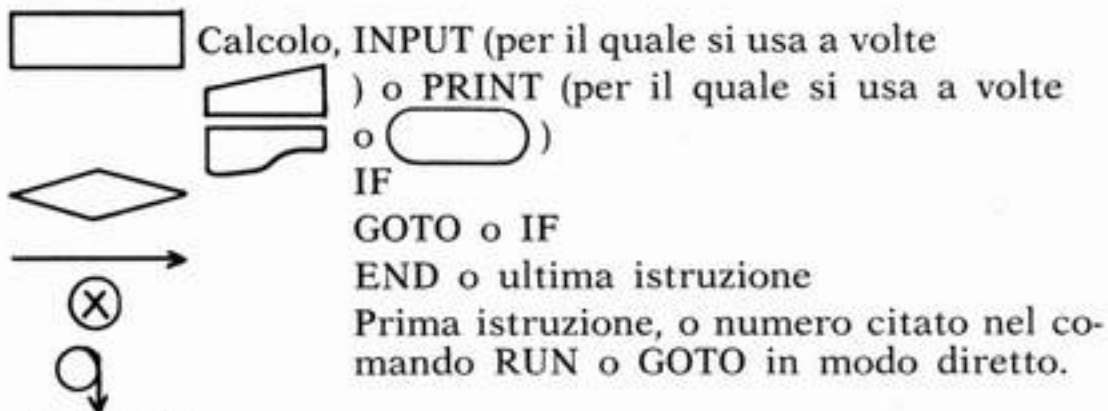
Esercizio 4.3

Calcolo, INPUT (per il quale si usa a volte) o PRINT (per il quale si usa a volte o)

IF

GOTO o IF

END o ultima istruzione

Prima istruzione, o numero citato nel comando RUN o GOTO in modo diretto.

Esercizio 4.4

PRINT INT(X * 10↑D)/10↑D

Esercizio 4.5 Due soluzioni:

1. fare la stampa in 65 e mettere 60 N = N + 1;
2. inizializzare N ad 1, il che si deve fare esplicitamente inserendo 10 N = 1.

Esercizio 4.7 Manca la stampa di una prima linea di titolo!

```
5 PRINT "NUMERO", "QUADRATO", "RADICE"
6 PRINT
```

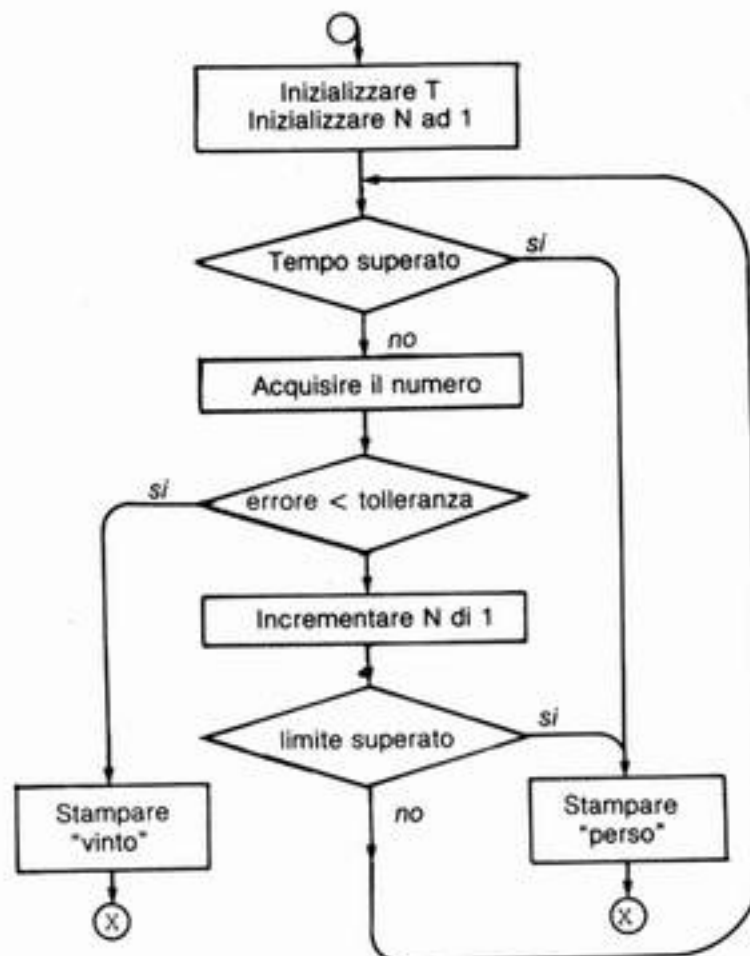
(Il 6 fa saltare una linea)

Esercizio 4.8 Unica istruzione modificata:

```
10 FOR N = 2 TO 10 STEP 2
```

Esercizio 4.9

10 : 1 3 5 7 valore finale 9
50 : 10 9 8 7 6 5 4 valore finale 3

Esercizio 4.11 2 minuti: (TI-T)/60 è il tempo trascorso in secondi; e accordiamo 120s.

Esercizio 4.13 Problema già risolto nell'istruzione 40:

60 PRINT "VINTO IN"; N; "TENTATIVI E"; INT((TI-T)/.6)/100; "SECONDI"

Esercizio 4.14

```
15 IF (TI-T)/60>120 GOTO 55
50 PRINT"NM. DI TENTATIVI AUTORIZZATI SUPERATO"
53 END
55 PRINT"TEMPO SUPERATO"
57 END
```

Esercizio 5.1

```
20 READ C:T=TI
25 IF C<>99999 GOTO 30
27 RESTORE:GOTO 20
100 GOTO 10
210 DATA 210,78.31,901.5,31,4.18,2.7,99999
```

Esercizio 5.2 Dopo il calcolo della media avremmo:

```
90 V=0
100 FOR I=1 TO N
110 V=V+(A(I)-M)↑2
120 NEXT I
130 V=V/(N-1)
```

Si può proporre una soluzione più elegante, che calcoli media e varianza nello stesso ciclo, e che tragga origine dalla seguente proprietà matematica:

$$\sum_i (A_i - M)^2 = \sum_i (A_i^2 - 2MA_i + M^2)$$

$$= \sum_i (A_i^2 - 2M \sum_i A_i + N * M^2$$

$$= 3 \sum_i A_i^2 - N * M^2$$

Ne deriva il programma:

Esercizio 5.2B

```
10 DIM A(N)
20 REM LETTURA DEGLI A
30 S=0:S2=0
40 FOR I=1 TO N
50 S=S+A(I):S2=S2+A(I)↑2
60 NEXT I
70 M=S/N:V=(S2-N*M↑2)/(N-1)
80 PRINT"MEDIA=";M,"VARIANZA=";V
```

Esercizio 5.3

```
10 DIM U(N),V(M)
20 REM LETTURA
30 UV=0
40 FOR I=1 TO N
50 UV =UV+U(I)*V(I)
60 NEXT I
```

Esercizio 5.5

```
10 REM NON CI OCCUPIAMO PER NIENTE
15 REM DELLE ENTRATE/USCITE
20 DIM A(N,N),B(N,N),C(N,N)
30 FOR I=1 TO N
40 FOR J=1 TO N
50 C(I,J)=0
60 FOR K=1 TO N
70 C(I,J)=C(I,J)+A(I,K)*B(K,J)
80 NEXT K:NEXT J:NEXT I
```

Esercizio 5.6

```
10 X$ = LEFT$(X$),4) + "A" + MID$(X$,6)
```

Esercizio 5.7

```
100 N=LEN(A$):P=LEN(B$)
110 FOR K=1 TO N-P+1
120 IF MID$(A$,K,P)=B$ GOTO 150
130 NEXT K
140 K=0
150 PRINT K
160 END
```

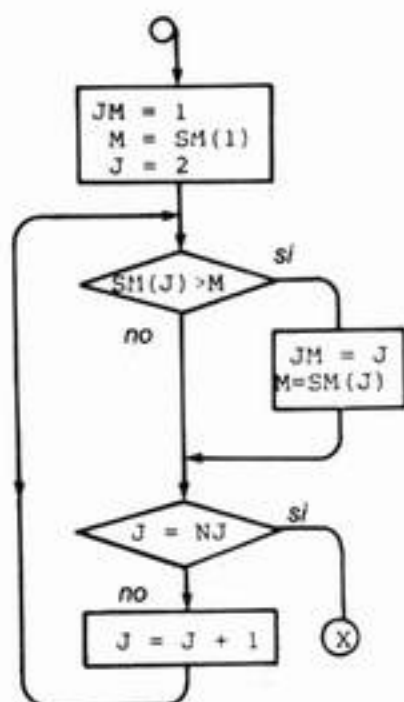
Quando avrete visto i sottoprogrami, capirete che è assai sensato sostituire 160 con 160 RETURN.

Esercizio 5.8

NC = LEN(STR$(A)) — 1

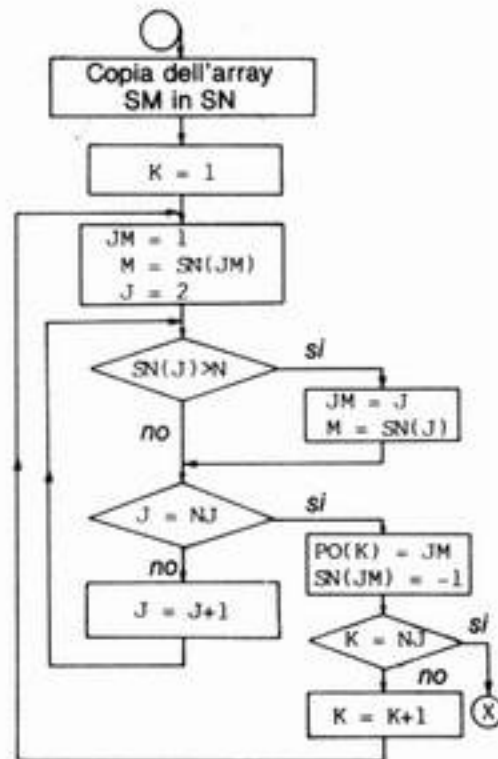
Esercizio 5.9

```
100 B$ = STR$(B): N = LEN(B$)
110 PRINT LEFT$(B$,N—3)
```

Esercizio 5.10

Si suppone in partenza che il massimo sia in posizione 1. Poi si percorre l'array SM dalla posizione 2 in poi. Se si trova un elemento maggiore del massimo supposto, si prende quell'elemento come nuovo massimo supposto.

Si può dunque aggiungere al programma B.10 la sequenza seguente:

```
210 PRINT:PRINT:DIM SM(NJ)
220 FOR J=1 TO NJ:SM(J)=0
230 FOR P=1 TO NP:SM(J)=SM(J)+SC(J,P)
240 NEXT P:SM(J)=SM(J)/NP:NEXT J
250 JM=1:M=SM(1)
260 FOR J=2 TO NJ
270 IF SM(J)<=M THEN 290
280 JM=J:M=SM(J)
290 NEXT J
300 PRINT"IL VINCITORE E  ";NOM$(JM)
310 PRINT"CON UN PUNTEGGIO MEDIO DI";M
```

Esercizio 5.11 I problemi di classificazione o di ordinamento sono tra i più frequenti. Si distinguono i problemi di ordinamento semplice dove, partendo da un array di elementi, si cerca di ottenere, alla fine del programma, lo stesso array, ma ordinato, con gli elementi ora crescenti o decrescenti; e i problemi di classificazione dove si lasciano al loro posto gli elementi dell'array di partenza, ma si forma un array ausiliario, detto array di puntatori PO(I), tala che PO(I) sia l'indice nell'array di partenza dell'elemento che merita di essere classificato Iesimo.
Il nome dell'Iesimo giocatore classificato è così NOM$(PO(I)).
Appena gli elementi sono ingombranti, è più interessante effettuare una classificazione piuttosto che un ordinamento semplice: i termini da spostare sono più piccoli.

Un metodo di classificazione semplice deriva dall'esercizio precedente. Supponiamo si voglia una classificazione per ordine di punteggio medio decrescente. PO(1) non è altro che il JM ottenuto in precedenza. PO(2) non è altro che la posizione del massimo successivo e così via.
Ma c'è un altro problema. Quando si cerca il massimo di una certa posizione, non bisogna riprendere un massimo ottenuto prima. Quindi, quando si trova un massimo, bisogna sostituire l'elemento con un valore minore di ogni valore possibile (qui —1) di modo che l'elemento non venga più ripreso. In tal caso, l'array SM verrà distrutto dalla

classificazione. Per evitarlo, si ricopia prima l'array SM in un array SN sul quale si lavorerà.
Ne deriva lo schema a blocchi qui di fronte che non è altro che la ripetizione di quello di 5.10 per ogni valore di K. K rappresenta la posizione ad un dato momento, nella classificazione.

```
250 DIM SN(NJ),PO(NJ)
260 FOR J=1 TO NJ:SN(J)=SM(J):NEXT
270 FOR K=1 TO NJ
280 JM=1:M=SM(JM)
290 FOR J=2 TO NJ
300 IF SN(J)<=M THEN 320
310 JM=J:M-SN(J)
320 NEXT J
330 PO(K)=JM:SN(JM)=-1
340 NEXT K
350 PRINT"ORDINAMENTO":PRINT
360 PRINT"POSIZIONE","GIOCATORE","PUNTEGGIO MEDIO":PRINT
370 FOR I=1 TO NJ
380 PRINT I,NOM$(PO(I)),SM(PO(I)):NEXT I
```

Esistono altri metodi di ordinamento. Uno dei più noti si chiama metodo del "bubble sort" (ordinamento a bolle). Si percorre l'array da ordinare confrontando ogni volta due elementi adiacenti. Se stanno nel senso giusto, li lasciamo, se stanno nell'ordine sbagliato, li scambiamo. Se, in un percorso, c'è stato almeno uno scambio, facciamo un nuovo percorso. Se non c'è stato alcuno scambio, vuol dire che, ormai, l'array è ordinato.
I metodi di ordinamento si applicano all'ordinamento alfabetico delle stringhe di caratteri, poiché IF A$ < B$ è vera se A$ precede B$. La sequenza seguente ordina per ordine alfabetico l'array dei nomi NOM$:

```
390 REM
400 ECH=0:REM INDICATORE DI SCAMBIO
410 FOR I=1 TO NJ-1
420 IF NOM$(I)<=NOM$(I+1) THEN 450
430 T$=NOM$(I+1):REM SALVAGUARDIA PER SCAMBIO
440 NOM$(I+1)=NOM$(I):NOM$(I)=T$:ECH=1
450 NEXT I
460 IF ECH=1 GOTO 400
470 PRINT:PRINT"ORDINAMENTO ALFABETICO"
480 FOR I=1 TO NJ:PRINT NOM$(I):NEXT I
```

Esercizio 6.1 Usiamo, ad esempio, i segni ╭ ╮ ╰ ╯ (Shift U,I,J,K) e istruzioni come 5 PRINT"... (poiché devono essere sopra a 10).

Esercizio 6.2 AABBAA (Si stampa AAAA, poi si inserisce BB in mezzo)

Esercizio 6.3 60 PRINT "$\bar{a}$*dddddddd* Rvs^c"; TI$: GOTO 60

Esercizio 6.5 POKE 1107,1: POKE 55379,5

Esercizio 6.6

```
300 PRINT "Clr"
310 FOR I = 1 TO 6:POKE 7951+I,128 + ASC (MID$(TI$,I))
320 POKE 55793 + I,0:NEXT I:GOTO 310
```

Esercizio 6.7

```
90 PRINT"RICOMINCIAMO?"
91 GET A$:IF A$="" GOTO 91
92 IF A$="S" GOTO 10
93 IF A$="N" THEN END
94 GOTO 91
```

Se battiamo qualsiasi carattere che non sia 0 o N, non ne viene tenuto conto: si ritorna al GET per avere il carattere successivo.

Esercizio 6.8 100 GET A$:IF A$ = "" GOTO 100
...seguito...

Esercizio 6.9 Le principali tappe della soluzione sono:

1. Decidere i caratteri da utilizzare per rappresentare il cucù. Proponiamo ♥ (cuore — codice schermo 83)
☒ (grande croce invertita, codice schermo 86 + 128 = 214).
2. Trovare gli indirizzi schermo per mettere il cucù nell'arco della porta. Vediamo, dagli ordini di impressione, che il cuore sarà in 17esima linea, 10ma colonna, dal che seguono gli indirizzi 1673, 1713 e 55945, 55985 (per il colore).

Ne consegue il programma C-4.

— **In 65**, si testa se si è ad un'ora esatta (vediamo l'uso delle operazioni logiche).
— **In 70**, si porta il cucù (di colore bianco grazie a 71).
— **In 75**, si lascia il cucù durante un certo intervallo (circa un decimo di secondo).
— **In 80**, si spegne il cucù (32 è il codice dello spazio).
— **In 85**, si mantiene il cucù spento durante un intervallo metà dell'intervallo precedente.

Se troviamo che l'apparizione del cucù è troppo fuggitiva, lo possiamo lasciare anche mentre il numero dei secondi sta a 01 scrivendo:

65 IF M$<>"00" OR S$<>"00" AND S$<>"01" GOTO 55

Si può anche far apparire il cucù ogni quarto d'ora. Manca soltanto un comando sonoro! (lo faremo nel seguito del capitolo).

PROGRAMMA C.4

```
5 PRINT"
10 PRINT"
15 PRINT"
20 PRINT"
25 PRINT"
30 PRINT"
35 PRINT"
40 PRINT"
45 PRINT"
50 PRINT"
52 PRINT
55 H$=LEFT$(TI$,2):M$=MID$(TI$,3,2):S$=RIGHT$(TI$,2)
60 PRINT" "H$" H "M$" M "S$" S "
65 IF M$<>"00" OR S$<>"00" AND S$<>"01" GOTO 55
70 POKE1673,83:POKE1713,214
71 POKE55945,1:POKE55985,1
75 FORI=1TO100:NEXT
80 POKE1673,32:POKE1713,32
85 FORI=1TO50:NEXT
90 GOTO 55
```

Esercizio 6.10

```
2000 PRINT""
2010 PRINT"
2015 PRINT"
2020 PRINT"
2025 PRINT"
2030 PRINT"
2035 PRINT"
2040 PRINT"
2045 PRINT"
2050 PRINT"
2055 PRINT"
2060 PRINT"
2065 PRINT"
2070 PRINT"
2075 PRINT"
2080 GOTO 2080
```

Esercizio 6.11

```
10 C$ = "Blkᶜ Redᶜ Cynᶜ Purᶜ Grnᶜ Bluᶜ Yelᶜ"
20 PRINT "Clr"
30 FOR I = 1TO505
40 K = INT(1 + 7 * RND(1)):K$ = MID$(C$,K,1)
50 PRINT K$; "*"; :NEXT
60 GOTO 60
```

L'istruzione principale è 40: si sorteggia un numero tra 1 e 7, ossia K. K$ è il Kesimo carattere di C$ che è precisamente la successione dei caratteri di colore (tranne bianco). Si stampano soltanto 999 caratteri poiché, con 1000, il sistema libererebbe linee bianche. Il 60 GOTO impedisce di stampare READY. Possiamo incorporare, in C$, i colori che si ottengono con 'C =' ed avere così una più grande scelta.

Esercizio 6.12

```
POKE 53280,0
POKE 53281,1 (quadro nero su sfondo bianco)
PRINT 'Blkᶜ' (le scritte saranno nere)
```

Non rimane che suonare la Marcia Funebre di Chopin, o la Danza Macabra di Saint-Saëns, cose che il C64 può benissimo fare.

Esercizio 6.13 Basta aggiungere nel programma C-5 o C-6:

```
1005 POKE 53281,6 : PRINT 'Yelᶜ'
```

6 fornisce uno sfondo blu (tenendo il quadro azzurro).

Esercizio 6.14

```
10 FOR I = 1024 TO 2023/POKE I,16 : NEXT
20 FOR I = 55296 TO 55615 : POKE I,2 : NEXT
30 FOR I = 5516 TO 55975 : POKE I,1 : NEXT
40 FOR I = 55976 TO 56295 : POKE I,6 : NEXT
50 GOTO 50
```

In 10, si riempie lo schermo di caratteri spazio in reverse.
È più facile che non la bandiera italiana perché, essendo le separazioni orizzontali, le zone di memoria da riempire dello stesso colore sono contigue.
Ovviamente, potremmo usare anche un metodo elementare con dei PRINT "controllo colore."

Esercizio 6.15 Si aggiungono le istruzioni (180 è modificato).

```
25 DATA159,9,159,9,159,9,216,12,216,12,107,14,107,14,63,19,
                                                47,16,216,12
26 DATA100,300,100,300,200,300,200,300,100,300
45 FOR N=1 TO 10:READ NL(N),NH(N):NEXT
46 FOR N=1 TO 10:READ ND(N):NEXT
180 PRINT"VINTO":GOSUB2000
2000 V=54296:L1=54272:H1=L1+1:W1=L1+4:A1=W1+1:S1=A1+1
2005 POKEV,10:POKEA1,31:POKES1,240
2010 FOR N=1 TO 10
2020 POKE L1,NL(N):POKE H1,NH(N):POKEW1,33
2030 FOR Z=1 TO ND(N):NEXT
2040 POKE W1,0
2050 FOR Z=1 TO 20:NEXT
2060 NEXT
2070 POKE W1,0:POKEV,0
2080 RETURN
```

NL e HH sono gli array delle frequenze e ND è quello delle durate.

Esercizio 6.16

```
5 PRINT""
6 V=54296:L1=54272:H1=L1+1:W1=L1+4:A1=W1+1:S1=A1+1
7 POKEV,15:POKEA1,31:POKES1,240
10 PRINT"
15 PRINT"
20 PRINT"
25 PRINT"
30 PRINT"
35 PRINT"
40 PRINT"
45 PRINT"
50 PRINT"
52 PRINT
55 H$=LEFT$(TI$,2):M$=MID$(TI$,3,2):S$=RIGHT$(TI$,2)
60 PRINT" "H$" H "M$" M "S$" S "
65 IF M$<>"00" OR S$<>"00" AND S$<>"01" GOTO 55
70 POKE1673,83:POKE1713,214
71 POKE55945,1:POKE55985,1
72 POKEL1,30:POKEH1,25:POKEW1,17
75 FORI=1TO200:NEXT
76 POKE W1,0
77 FORI=1TO40:NEXT
80 POKE1673,32:POKE1713,32
82 POKEL1,48:POKEH1,28:POKEW1,17
85 FORI=1TO100:NEXT
86 POKE W1,0
87 FORI=1TO200:NEXT
90 GOTO 55
```

Esercizio 6.17 GIOCO "SIMON"

```
10 DATA195,16,209,18,31,21,96,22,30,25,49,28,165,31,135,33
15 V=54296:L1=54272:H1=L1+1:W1=L1+4:A1=W1+1:S1=A1+1
20 FOR N=1 TO 8:READ NL(N),NH(N):NEXT
25 POKE V,10:POKEA1,31:POKES1,240
30 PRINT"♥"
40 FOR I=1 TO 4
50 NS(I)=INT(1+S*RND(1)):NEXT
60 FOR I=1 TO 4
70 POKE 53281,NS(I)-1
80 POKE L1,NL(NS(I)):POKE H1,NH(NS(I)):POKE W1,17
90 FOR Z=1 TO 400:NEXT
100 POKE W1,0:NEXT
110 T=0:POKE 53281,6
120 FOR I=1 TO 4
130 GET A$:IF A$=""GOTO130
140 SN(I)=ASC(A$)-48
150 POKE 53281,SN(I)-1:POKEL1,NL(NS(I)):POKEH1,NH(SN(I)):
                                                   POKEW1,17
160 FOR Z=1TO 400:NEXT
170 POKE W1,0:NEXT
180 BENE=-1:POKE 53281,6:PRINT"♥"
190 FOR I=1 TO 4
200 BENE=BENE AND (NS(I)=SN(I)) :NEXT
210 IF BENE THEN PRINT"VINTO":GOTO 240
220 IF T=0 THEN PRINT"RIPROVA":T=1:GOTO 120
230 PRINT"PERSO! ERA"
235 FOR I=1 TO 4:PRINT NS(I);:NEXT:PRINT
240 INPUT "RICOMINCIAMO ";A$
250 IF A$="SI" GOTO 30
```

NS è la sequenza preparata dal computer, mentre SN è quella proposta dal giocatore. Notate la variabile logica BENE e la sua gestione. Ovviamente, potete apportare varianti: numero di note per sequenza e numero di tentativi variabili, possibilità di risentire la sequenza, trattamento del caso in cui un giocatore batte su di un tasto diverso da quelli da 1 a 8.

Esercizio 6.18

```
5 V=54296:L1=54272:H1=L1+1:W1=L1+4:A1=W1+1:S1=A1+1
10 POKEA1,31:POKES1,240:POKEL1,50:POKEH1,0:POKEW1,129
20 FOR A=4 TO 15
30 POKE V,A:POKEW1,129
40 FOR Z=1 TO 1500:NEXT:NEXT
```

```
50 FOR Z=1 TO 500:NEXT
60 FOR A=15 TO 4 STEP-1
70 POKE V,A:POKEW1,129
80 FOR Z=1 TO 1500:NEXT:NEXT
90 POKE V,0
```

Esercizio 7.1 Basta mettere nel ciclo un secondo comando di stampa per la seconda funzione, iniziando con una "ā" per venire sulla stessa linea.

```
10 PRINT "⌉"
20 FOR X=0 TO 2*π STEP 2*π/22
30 N1=1+INT(36*(SIN(X)+1)/2)
35 N2=1+INT(36*(COS(X)+1)/2)
40 PRINT TAB(N1);"*"
45 PRINT "⌉";TAB(N2);"+"
50 NEXT X
60 GOTO 60
```

Il secondo grafico si scrive con dei +. Abbiamo ridotto l'intervallo di scrittura a 23 linee (ne segue il 22 dell'istruzione 20) e l'intervallo di variazione a 37 (ne segue il 20 in 30 e 35) affinché i grafici tengano nello schermo.

Esercizio 7.2 Si pongono problemi: il primo è quello della scala: gli effettivi sono tutti abbastanza vicini (da 80 a 120), quindi si devono dilatare le differenze, ma comunque tenere un certo termine costante.
Siccome il numero della classe e l'effettivo sono scritti ad inizio linea, la scrittura inizia in colonna 11.
Proponiamo la formula di riduzione di scala N = 1/2(C(I) — 60) che fa variare N da 10 a 30. Peraltro, siccome ?"Rvs^c"; TAB(N); "Off^c" non fa disegnare N spazi, bisogna fare ?"Rvs^c"; :FOR K = 1 TO N: ?"Sp"; :NEXT: ?"Off^c".
Ne consegue la fine del programma:

```
90 PRINT "CL ";" EFF"
100 FOR I=1 TO 10:N=INT((C(I)-60)/2)
110 PRINT I;C(I),"▣";
120 FOR K=1 TO N:PRINT" ";:NEXT K
130 PRINT "■":NEXT I
```

Esercizio 7.3

```
4 POKE 52,48:POKE56,48:CLR
5 POKE56334,PEEK(56334)AND254
6 POKE1,PEEK(1)AND251
10 POKE1,PEEK(1)AND251
20 FOR I=0 TO 2047
```

```
30 POKE G2+I,PEEK(G1+I):NEXT
35 POKE1,PEEK(1)OR4
36 POKE56334,PEEK(56334)OR1
40 DATA0,0,59,126,108,108,126,59
45 R=0
50 FOR L=0 TO 7:READ A:POKE G2+8*R+L,A:NEXT
60 POKE 53272,(PEEK(53272)AND240)+12
70 PRINT"[illegible] = ALFA"
```

Esercizio 7.4 Così come A = A, OR 2↑B forza un bit 1 nella posizione B di A = A AND (255—2↑B) forza uno 0 per cui;

```
2000 K = X AND 7 : L = Y AND 7 : Q = P + L : Z = Z/(7-K)
2010 POKE Q, PEEK(Q) AND (255-Z) : RETURN
```

Esercizio 7.5

PROGRAMMA D-2 — TELESCHERMO

```
10 POKE53272,PEEK(53272) OR 8
20 POKE52,32:POKE56,32:CLR
30 E=8192:CO=1024
40 FORI=CO TO CO+999:POKEI,1:NEXT
45 FORI=E TO E+7999:POKEI,0:NEXT
50 POKE53265,PEEK(53265) OR 32
60 X=0:Y=0
70 GOSUB 900:GOSUB 1000
80 A=PEEK(203):B=PEEK(653):IF A=64 GOTO 80
100 IF A=7 THEN IX=0:IY=1:GOTO 200
110 IF A=2 THEN IX=1:IY=0:GOTO 200
120 IF A=51 GOTO 350
125 IF A=0 THEN IX=-1:IY=0:GOTO 200
130 IF A=4 THEN IX=-1:IY=-1:GOTO 200
135 IF A=5 THEN IX=1:IY=-1:GOTO 200
140 IF A=6 THEN IX=-1:IY=1:GOTO 200
145 IF A=3 THEN IX=1:IY=1:GOTO 200
150 GOTO80
200 LX=(IX+1)*159.5:LY=(IY+1)*99.5:IF (X=LX)OR(Y=LY) GOTO80
210 IF B AND 4 GOTO 230
220 GOSUB 2000
230 X=X+IX:Y=Y+IY:GOSUB900:GOSUB1000:GOTO 80
350 IF B AND 1 THEN RUN
355 IF B AND 2 GOTO 60
360 IX=0:IY=-1:GOTO 200
900 XM=INT(X/8):YM=INT(Y/8):P=E+320*YM+8*XM:RETURN
1000 K=X AND 7:L=Y AND 7:Q=P+L
1010 POKE(Q),PEEK(Q) OR2↑(7-K):RETURN
2000 K=X  AND 7:L=Y AND 7:Q=P+L:Z=2↑(7-K)
2010 POKE Q,PEEK(Q) AND (255-Z):RETURN
```

Per far apparire il carattere di codice schermo R nella maglia di coordinate XM,YM, basta copiare gli 8 bytes voluti del generatore di caratteri negli indirizzi da P a P + 7.

Esercizio 7.6 FOR L = 0 TO 7 : POKE P + L, PEEK(6 + 8 * R + L): NEXT (non dimenticate le precauzioni rappresentate dalle istruzioni 5,6,35,36, pagina 101).
È per facilitare ciò che la visualizzazione alta risoluzione del C64 è organizzata in maglie, il che sembra strano a primo acchito.

Esercizio 7.7

```
5 PRINT"▚▮▮▮▮▮▮▮▮▮▮▮▮"
10 PRINT"●";:D=6:K=1:S=1
20 PRINT"▮▮ ●";:K=K+1
30 GOSUB 1000:IF K<40 GOTO 20
40 PRINT"▮▮ ▮▮▮●";:K=K-1
50 GOSUB 1000:IF K>1 GOTO 40
60 D=D+S:IF D=10 THEN S=-1
70 IF D=1 THEN S=1
80 GOTO 20
1000 T=TI
1010 IF TI-T<D GOTO 1010
1020 RETURN
```

Esercizio 7.8

```
5 PRINT"▚▮▮▮▮▮▮▮▮▮▮▮▮"
10 PRINT"●";:D=6:K=1
20 PRINT"▮▮ ▮▮▮●";:K=K+1
30 GOSUB 1000:IF K<25 GOTO 20
40 PRINT"▮▮ ▮▚●";:K=K-1
50 GOSUB 1000:IF K>1 GOTO 40
60 GOTO 20
1000 T=TI
1010 IF TI-T<D GOTO 1010
1020 RETURN
```

Esercizio 7.9

```
5 PRINT"▚"
10 PRINT"●";:D=6:K=1
20 PRINT"▮▮ ▮▮▮●";:K=K+1
30 GOSUB 1000:IF K<25 GOTO 20
40 PRINT"▮▮ ▮▚●";:K=K-1
50 GOSUB 1000:IF K>1 GOTO 40
60 GOTO 20
1000 T=TI
1010 IF TI-T<D GOTO 1010
1020 RETURN
```

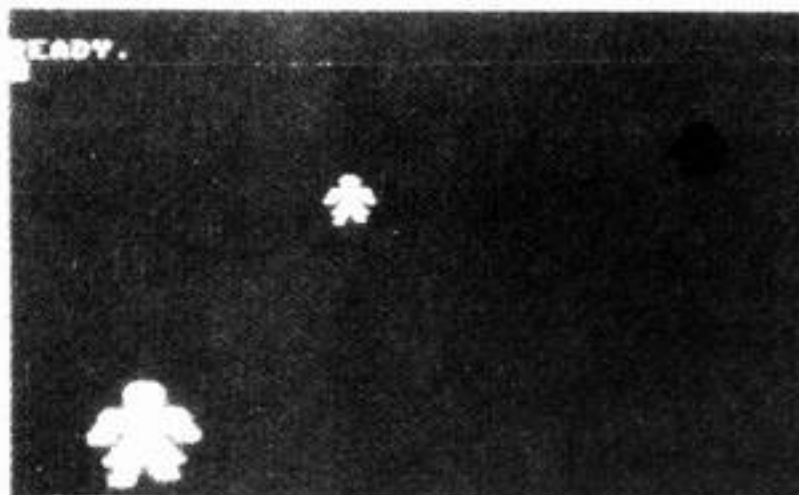
Esercizio 7.10 Basta avere come inizio di programma:

```
10 POKE53269,1:POKE53287,0:POKE53249,100
20 FOR X=24 TO 321
30 POKE 53248,X AND 255:POKE 53264,-(X>255)
40 NEXT
50 FOR X=321 TO 24 STEP -1
60 POKE 53248,X AND 255:POKE 53264,-(X>255)
70 NEXT:GOTO 20
```

Provate ad aggiungere STEP 5(o 10) nell'istruzione 20 e osservate la velocità.

Esercizio 7.11 L'istruzione 10 fa puntare gli OGS 1 e 2 alla stessa zona che l'OGS 0: avranno quindi la stessa forma. L'istruzione 50 dilata l'OGS 1 (il bianco) nei due sensi.

```
10 POKE53277,2:POKE53271,2
20 POKE53269,7:POKE53287,0:POKE53288,1:POKE53289,4
30 POKE53249,100:POKE53253,120:POKE53251,200
40 POKE53248,30:POKE53252,150:POKE53250,50:POKE53264,1
50 POKE53277,2:POKE53271,2
60 END
```

Ecco ciò che viene visualizzato:

Esercizio 8.1 Si mandano alternativamente 0 V e 5 V sull'altoparlante (segnale a merli).
Si ottiene dunque un suono.
Il programma citato fornisce il merlo più rapido che si possa ottenere in Basic. Ora, la frequenza così ottenuta è dell'ordine di 100 a 200 Hz.
Quindi, per fare musica con un C64 con questo metodo, bisognerà ricorrere al linguaggio macchina. Ma, il C64, l'abbiamo visto, dispone di molto meglio per creare effetti sonori.

BIBLIOGRAFIA

Libri

Programmer en Basic
di Michel Plouin (Edizioni del P.S.I).
Jeux, trucs et comptes pour PET/CBM
di Michel Benelfoul (Edizioni del P.S.I.).

Nell'attesa dei prossimi libri della serie sul C64, troverete informazioni sulle periferiche Commodore in:

La pratica del VIC - Volume 1°
di Daniel-Jean David (Edizioni del P.S.I).
Clefs pour le VIC
di Daniel-Jean David (Edizioni del P.S.I.).

In inglese:

32 Basic programs for the PET computer
di Tom Rugg e Phil Feldman (Dilithium Press).

In italiano:

Impariamo a programmare in Basic con il VIC/CBM
di Rita Bonelli (Gruppo Editoriale Jackson).
Commodore 64 - IL BASIC (Gruppo Editoriale Jackson).

Riviste

L'Ordinateur Individuel
39, rue de la Grange aux Belles - 75010 Paris.
La Commode, le magazine des ordinateurs Commodore
28, rue Vicq-d'Azir - 75010 Paris.

In inglese:

VIC Computing.

La scoperta del Commodore 64

1. introduzione al Basic

Il Commodore 64 è un computer che permette applicazioni professionali e giochi al tempo stesso. Questo libro di introduzione copre entrambi gli aspetti e non richiede conoscenze già acquisite. Dopo una introduzione costituita da richiami generali sull'informatica, viene presentato il Basic in modo semplice e progressivo. La scoperta del linguaggio è condotta costruendo dei programmi per approfondimenti successivi nel corso dei quali i concetti nuovi vengono introdotti con naturalezza. Sono trattati in particolare i punti forti del Commodore 64 e cioè la grafica, il suono, il colore, l'alta risoluzione e gli sprite.

I.S.B.N. 88.7688.**200**.6